5-23-2003

A flexible hardware architecture for 2-D discrete wavelet transform: design and FPGA implementation

Richard Carbone

A Flexible Hardware Architecture for 2-D Discrete Wavelet Transform: Design and FPGA Implementation

by

Richard L. H. Carbone

May 23, 2003

A thesis submitted in partial fulfillment of
the requirements for the degree of

Masters of Science in
Computer Engineering

Rochester Institute of Technology

Approved by:

Principal Advisor: _____
Dr. Andreas Savakis, Associate Professor and Department Head

Committee Member: _____
Dr. Marcin Lukowiak, Visiting Assistant Professor

Committee Member: _____
Dr. Greg Semeraro, Assistant Professor

Release Permission Form

Rochester Institute of Technology

A Flexible Hardware Architecture for 2-D Discrete Wavelet Transform: Design and FPGA Implementation

I, Richard Carbone, hereby grant permission to the Wallace Library of the Rochester Institute of Technology to reproduce my thesis in whole or in part. Any reproduction will not be for commercial use or profit.

Richard L. H. Carbone

5/23/2003

Date

Abstract

The Discrete Wavelet Transform (DWT) is a powerful signal processing tool that has recently gained widespread acceptance in the field of digital image processing. The multiresolution analysis provided by the DWT addresses the shortcomings of the Fourier Transform and its derivatives. The DWT has proven useful in the area of image compression where it replaces the Discrete Cosine Transform (DCT) in new JPEG2000 and MPEG4 image and video compression standards. The Cohen-Daubechies-Feauveau (CDF) 5/3 and CDF 9/7 DWTs are used for reversible lossless and irreversible lossy compression encoders in the JPEG2000 standard respectively.

The design and implementation of a flexible hardware architecture for the 2-D DWT is presented in this thesis. This architecture can be configured to perform both the forward and inverse DWT for any DWT family, using fixed-point arithmetic and no auxiliary memory. The Lifting Scheme method is used to perform the DWT instead of the less efficient convolution-based methods. The DWT core is modeled using MATLAB and highly parameterized VHDL. The VHDL model is synthesized to a Xilinx FPGA to prove hardware functionality. The CDF 5/3 and CDF 9/7 versions of the DWT are both modeled and used as comparisons throughout this thesis. The DWT core is used in conjunction with a very simple image denoising module to demonstrate the potential of the DWT core to perform image processing techniques.

The CDF 5/3 hardware produces identical results to its theoretical MATLAB model. The fixed-point CDF 9/7 deviates very slightly from its floating-point MATLAB model with a ~59dB PSNR deviation for nine levels of DWT decomposition. The execution time for performing both DWTs is nearly identical at ~14 clock cycles per image pixel for one level of DWT decomposition. The hardware area generated for the CDF 5/3 is ~16,000 gates using only 5% of the Xilinx FPGA hardware area, 2.185 MHz maximum clock speed and 24 mW power consumption. The simple wavelet image denoising techniques resulted in cleaned images up to ~27 PSNR.

Acknowledgements

The author would like to acknowledge Dr. Andreas Savakis, Dr. Marcin Lukowiak, and Dr. Greg Semeraro for serving as advisors on the thesis committee. The author would also like to thank family and friends for their support.

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

The digital multimedia revolution is upon us. The exchange of information is rapidly moving away from the traditional analog realm and into the digital arena fostered by the increasing availability of the internet in homes and at work. The speed and ease of which digital media is transferred and manipulated makes it an attractive alternative to conventional analog media such as audio and video. An example of digital media that has received a lot of attention in recent years is digital images. Digital image processing techniques can be used to compress, reduce noise, or even understand information present in digital images making this format even more desirable.

The Discrete Wavelet Transform (DWT) is a signal processing technique that is beginning to show promise in the field of digital image processing. The new JPEG2000 and MPEG4 still image and video compression standards are based upon the DWT and are shown to produce superior results over their previous incarnations that do not use the DWT [7]. The DWT has potential for many other applications to digital images other than compression such as noise reduction [12]. A hardware DWT core could be integrated into digital camera or scanners to perform image processing inside the device such as image compression to increase the amount of images that can be stored internally. Despite the benefits a hardware DWT core could provide to these applications there are very few hardware implementations of the DWT commercially available [19, 20].

A flexible hardware architecture for performing the DWT on a digital image is presented in this thesis. This architecture uses a variation of the lifting scheme technique that provides significant advantages over the both the standard lifting scheme-based DWT and convolution-based DWT, such as smaller memory requirements, fixed-point arithmetic instead of more costly floating-point, and less arithmetic computations [9]. In addition, the architecture is flexible in that it can be configured to perform many different variations of the DWT. For example the JPEG2000 still image compression standard uses the Cohen-Daubechies-Feauveau (CDF) 5/3 and CDF 9/7 DWT for lossless and lossy image compression respectively [17]. The architecture presented in

this thesis is flexible in that it allows for the hardware for either of these two DWTs to be generated automatically.

The DWT core is modeled using MATLAB and VHDL. The VHDL model is synthesized to a Xilinx FPGA to prove hardware functionality. The CDF 5/3 and CDF 9/7 versions of the DWT are both modeled and used as comparisons throughout this thesis. The DWT core is used in conjunction with a very simple image denoising module to demonstrate the potential of the DWT core to perform image processing techniques.

Following the introduction provided here in Chapter 1, an introduction to the discrete wavelet transform is provided through a brief discussion of signal processing techniques in Chapter 2. Techniques for performing the discrete wavelet transform are explored. The application of the discrete wavelet transform to digital image processing and current implementations are also discussed. Chapter 3 describes the unique features of the DWT core developed in this thesis. The design of the DWT core and its implementation is discussed in detail. Chapter 4 discusses the verification of the DWT core. A performance analysis of the DWT core is given along with results from hardware synthesis. Finally the results from applying wavelet denoising to images are discussed. Chapter 5 provides concluding remarks and explores various recommendations for future work in the area of the discrete wavelet transform, particularly the DWT core designed for this thesis. Following Chapter 5 are the reference section and appendix. Appendix A contains images transformed using the discrete wavelet transform, Appendix B contains images and results from wavelet image denoising, Appendix C, D, and E contain MATLAB, VHDL and C source code respectively.

Chapter 2. Background

This chapter introduces the discrete wavelet transform through a brief discussion of signal processing techniques. Techniques for performing the discrete wavelet transform are explored. The application of the discrete wavelet transform to digital images and current implementations are also discussed.

2.1. Wavelet Motivation

Signals are omni-present in the real world. Signals are the vehicle for delivering information such as sound and images. Signals represent a value or amplitude that varies depending on its location within a particular domain. Some signals occur in the time domain; that is the amplitude of the signal varies with time. Other signals occur in the spatial domain; that is the amplitude of the signal varies depending on its spatial location. An audio signal is an example of a time-domain signal having an amplitude that varies with time. A digital image is an example of a spatial-domain signal having an amplitude, or intensity that varies depending on the spatial location within the image.

Signal processing involves the conditioning of raw data signals into a form that is more suitable for their intended application. Mathematical transforms serve as a powerful tool used to extract information which is not readily available from the raw signal by converting a signal from one domain to another. Often information that is embedded in one domain, such as the time domain, is more apparent in another domain, such as the frequency domain. One example from [1] is the Electrocardiography (ECG) signal. "The typical shape of a healthy ECG signal is well known to cardiologists. Any significant deviation from that shape is usually considered to be a symptom of a pathological condition. This pathological condition, however, may not always be quite obvious in the original time-domain signal. Cardiologists usually use the time-domain ECG signals which are recorded on strip-charts to analyze ECG signals. Recently, the new computerized ECG recorders/analyzers also utilize the frequency information to decide whether a pathological condition exits. A pathological condition can sometimes be diagnosed more easily when the frequency content of the signal is analyzed" [1]. A mathematical tool for obtaining the frequency domain information from a time-domain signal is the Fourier Transform.

2.1.1. Fourier Transform

In 1882 the French mathematician J. Fourier showed that any periodic function can be decomposed into an infinite sum of periodic complex exponential functions, or sinusoids. After many years Fourier's ideas were generalized to non-periodic functions and finally to periodic or non-periodic discrete time signals resulting in the Fourier Transform (FT). The FT converts a time-domain signal into the frequency domain thus providing information as to which frequency components are present in a signal. The FT provides perfect resolution in the frequency domain; in other words the exact frequencies present in the signal are determined. The FT also provides perfect time resolution in the time domain as the value of the signal at every instant of time is known.

The shortcomings of the FT are that while the frequencies present in a signal can be determined with perfect resolution, there is zero resolution in the time domain as the time location of these frequencies are unknown. Hence the FT is sufficient for stationary signals where frequency does not vary with time, but for non-stationary signals whose frequency does vary with time, simply knowing which frequency components exists may not be sufficient. A possible solution to finding both the frequency component and where it occurs in time is the Short Term Fourier Transform. [2]

2.1.2. Short Term Fourier Transform

The FT is suitable only for stationary signals but not for non-stationary signals. The solution is to find portions of a non-stationary signal that are stationary. The Short Term Fourier Transform (STFT) does exactly this by segmenting a non-stationary signal into sections, or windows, and treating each window as a stationary signal. The FT is then performed on each window of stationary signals. A time-frequency representation can now be obtained from the signal since frequency information for each window has a location in time.

The shortcomings of the STFT are rooted in the Heisenberg Uncertainty Principle. Originally applied to momentum and location of moving particles, this principle can be applied to time-frequency information of a signal. Applied to time-domain signals this principle states that one cannot know the exact time-frequency representation of a signal. However, only the time

intervals in which certain bands of frequencies exist can be known providing imperfect resolution in both the frequency and time domains. Narrow windows provide good time resolution, but poor frequency resolution. Conversely, wide windows provide good frequency resolution, but poor time resolution. In addition, wide windows may no longer contain stationary signals defeating the purpose of the STFT. Whereas the kernel in the FT is a window of infinite length providing perfect frequency resolution and zero frequency resolution, the kernel in the STFT is a window of finite length yielding imperfect resolution in both cases. The time and frequency resolution problems are results of a physical phenomenon and exist regardless of the transform used, however, it is possible to analyze a signal using a different approach, called Multiresolution Analysis. [2, 3]

2.2. Multiresolution Analysis and the Wavelet Transform

Multiresolution analysis (MRA) analyzes signals at multiple frequencies yielding different resolutions at each frequency. At higher frequencies good time resolution is obtained at the expense of poorer frequency resolution. Conversely, at lower frequencies there is good frequency resolution, but poor time resolution. The power of MRA lies in the fact that certain information may go undetected at one resolution, but may be readily apparent at another resolution. [2, 4]

The STFT divided a non-stationary signal into a finite number of stationary signals, windows, for analysis. As discussed earlier, the resolution problems arose from the manner in which the original non-stationary signal was segmented. Using smaller windows yielded poor frequency resolution and using larger segments provided poor time resolution. One solution is to use a fully scalable, modulated window that is shifted along the signal at every position to extract frequency information. This process would then be repeated many times with slightly shorter or longer windows for each new repetition until every size window has been applied to every position of the signal. The end result will be a collection of time-frequency representations of a signal with different resolutions, and thus MRA is performed. The Continuous Wavelet Transform is performed in exactly this manner. [3]

2.2.1. Continuous Wavelet Transform (CWT)

The Continuous Wavelet Transform (CWT) is the most recent solution to overcome the shortcomings of the FT and STFT providing perfect resolution in both the time domain and frequency domain.

The term "wavelet" literally means "small wave". A wavelet is a function of finite length (small) and which is oscillatory (wave) having an average value, integral, of zero. These are the most important properties of a wavelet as they satisfy the admissibility and regularity conditions required for decomposition (analysis) and reconstruction (synthesis) of a signal without loss of information. Further information is provided in [3] regarding the details of the admissibility and regularity conditions. Whereas basis functions for the FT, and hence the STFT, are sinusoids (the FT decomposes a signal/function into a series of sinusoids), the basis functions for the CWT are known as "baby wavelets". More specifically, the CWT decomposes a signal or function into a series of baby wavelet functions. These baby wavelets are derived from a single prototype wavelet via dilations or contractions (scaling) and translations (shifts). This prototype wavelet is aptly named the "mother wavelet". An example of a mother wavelet and derived wavelets are shown in Figures 2.1, 2.2, and 2.3.

A baby wavelet $\psi_{s,\tau}(t)$ is derived from the mother wavelet $\psi(t)$ by varying scaling and translation parameters s and τ respectively as shown in equation 2.1. The $\frac{1}{\sqrt{s}}$ is for energy normalization across the different scales.

$$\psi_{s,\tau}(t) = \frac{1}{\sqrt{s}}\psi\left(\frac{t-\tau}{s}\right) \qquad (2.1)$$

The CWT is performed by multiplying the signal to be analyzed by all the baby wavelets having the same scale (s) but different translations (τ). In doing so, the scale information for every value in time is obtained. Scale information is considered to be inversely related to the frequency information as a larger scale value refers to lower frequency and vice-versa. The process is then repeated using dilated or contracted baby wavelets at every scale (s) until all scale-translation (τ,s) combinations of baby wavelets are applied to the signal thus resulting in a time-scale MRA of the signal. Note that the wavelet MRA is in time-scale resolution whereas the discussion

earlier referred to a time-frequency resolution. [3, 5] The CWT $\gamma(s,\tau)$ of a function $f(t)$ is defined in equation 2.2. * denotes complex conjugation. The s and τ parameters represent the new scale and translation scales respectively.

$$\gamma(s,\tau) = \int f(t)\psi^*_{s,\tau}(t)dt \qquad (2.2)$$

For completeness the inverse CWT transform is defined in equation 2.3.

$$f(t) = \int\int \gamma(s,\tau)\psi_{s,\tau}(t)d\tau ds \qquad (2.3)$$

Figure 2.1: Example of a Mother Wavelet

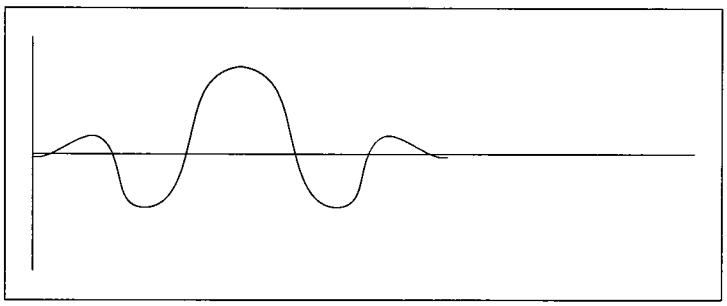


Figure 2.2: Example of Scaled Baby Wavelet

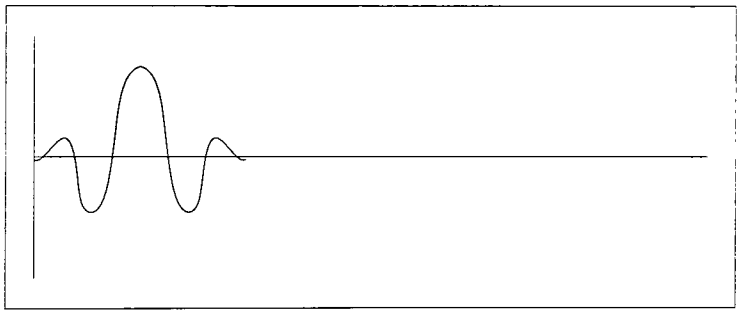
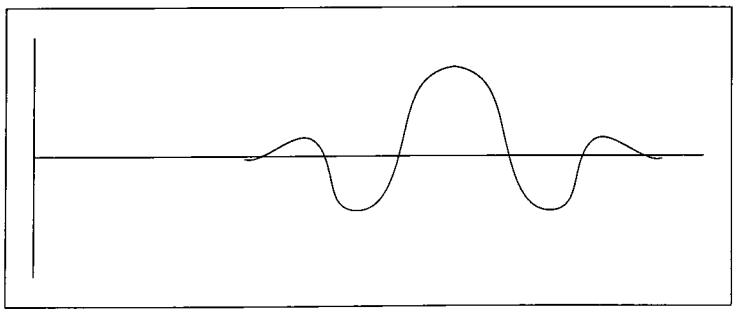


Figure 2.3: Example of Translated Baby Wavelet



The CWT addresses the limited time and frequency resolution shortcoming of the STFT by providing frequency (scale) information of a signal at many different resolutions hence providing a MRA of a signal.

Since computers perform almost all calculations and processing of signals in the real world, there is a concern about how practical the CWT is to implement. There are three properties of the CWT that make it difficult to use. First, the CWT is performed by continuously shifting a continuously scalable function over a signal and performing calculations between the two. The second problem is that there is an infinite number of wavelets in the CWT. The third problem is that for most functions the wavelet transforms have no analytical solutions and can be calculated only numerically or by an optical analog computer. The Discretized Continuous Wavelet Transform can be used to perform the CWT using computers and thus provide the wavelet series of a signal, however, this is only a sampled version of the CWT and is still highly redundant and therefore inefficient. As a result the Discrete Wavelet Transform was developed to address these issues and make wavelet processing more practical. [3]

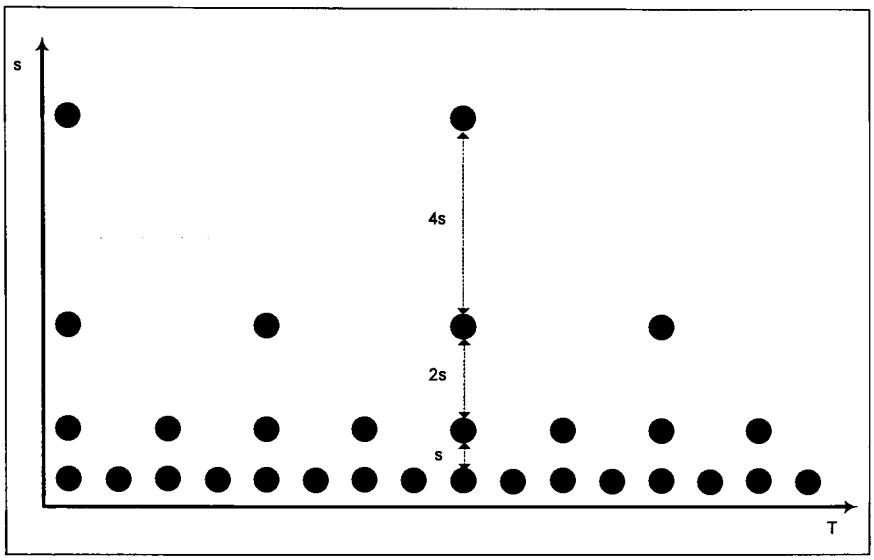
2.2.2. Discrete Wavelet Transform (DWT)

The Discrete Wavelet Transform (DWT) requires a discrete mother wavelet since the computation complexity of performing analysis of a signal with a continuous wavelet as in the CWT is not efficient. Discrete wavelets can only be scaled and translated in discrete steps as they are not continuously scalable or translatable. The representation for the new discretized wavelet is shown in equation 2.4, j and k are integers and $s_0 > 1$ is a fixed dilation step. The translation factor τ_0 is dependent upon s_0.

$$\psi_{j,k}(t) = \frac{1}{\sqrt{s_0^j}} \psi\left(\frac{1 - k\tau_0 s_0^j}{s_0^j} \right) \qquad (2.4)$$

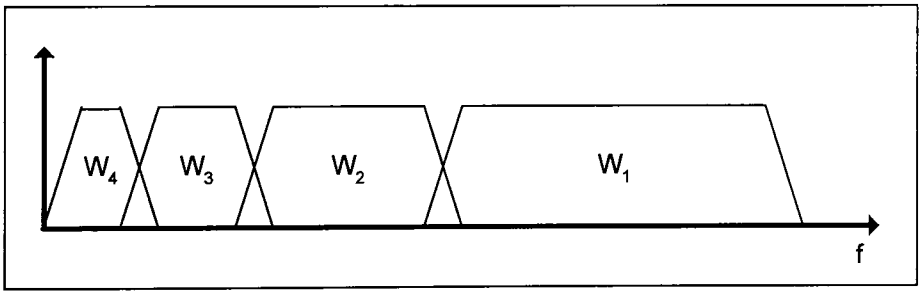
The effect of discretizing the wavelet is that the time-scale space is now sampled at discrete intervals. A value of $s_0 = 2$ and $\tau_0 = 1$ are usually chosen so that the sampling of the frequency and time axes correspond to dyadic sampling which is illustrated in Figure 2.4. One reason for the choice of dyadic sampling is that it is a very natural choice for computers [3].

Figure 2.4: Dyadic Sampling



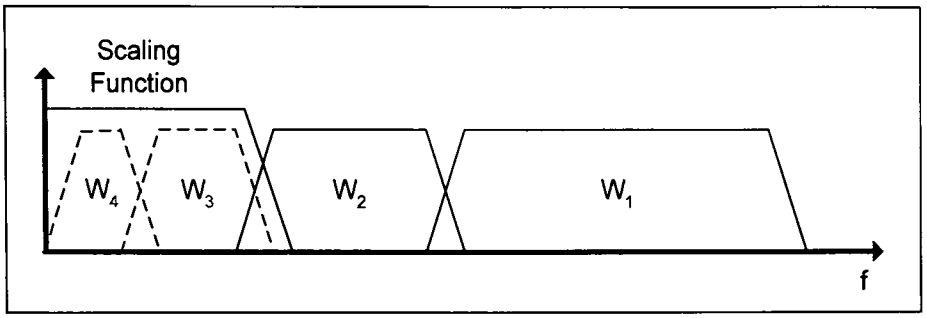
Even with a discrete wavelet the wavelet transform requires an infinite number of scalings and translations of the mother wavelet, however, this is not possible with a discrete algorithm such as the DWT. In order to provide good coverage of the signal spectrum using a finite number of wavelets the scaling factor of 2 is used and by doing so each wavelet will touch each other as shown in Figure 2.5. [3]

Figure 2.5: Subband Decomposition without Scaling Function



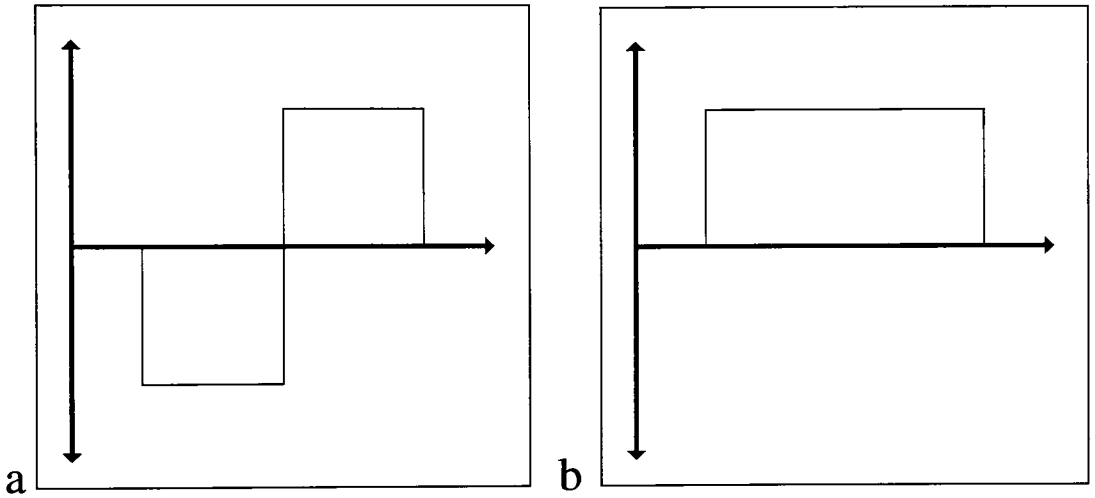
It is impossible to cover the spectrum all the way down to zero as the spectrum is continually halved and never reaches zero. As a result, Mallat introduced a scaling function which effectively "corks" the remaining spectrum thus requiring only a finite number of wavelets. As shown in Figure 2.6, this "cork" fills the void with a low-pass spectrum commonly referred to as the scaling filter [3]. Similarly the translation factor is chosen to be 2 to provide complete coverage of the time range of the signal.

Figure 2.6: Subband Decomposition with Scaling Function



An example of a Haar family wavelet and scaling functions are shown in Figure 2.7a and b respectively.

Figure 2.7: Haar Family Wavelet (a) and Scaling Function (b)

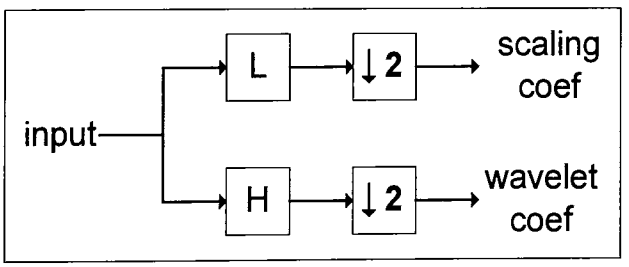


Now that the continuous issues of the CWT have been addressed, a method is required to perform the DWT on a signal. In 1976 Croisier, Esteban and Galand used a special type of analysis-synthesis system known as a Quadrature Mirror Filter (QMF) filter bank to perform an analysis of speech signals and named their analysis scheme "subband coding". A technique very similar to subband coding, called pyramidal coding, was defined by Burt in 1983. This technique is also known as MRA mentioned earlier. In 1989 Vetterli and Le Gall modified the subband coding scheme and thus removed the existing redundancy in the in the pyramidal coding scheme. [6]

The wavelet transform is computed by changing the scale of an analysis window (mother wavelet) and shifting this window in time across the signal to obtain the time-scale

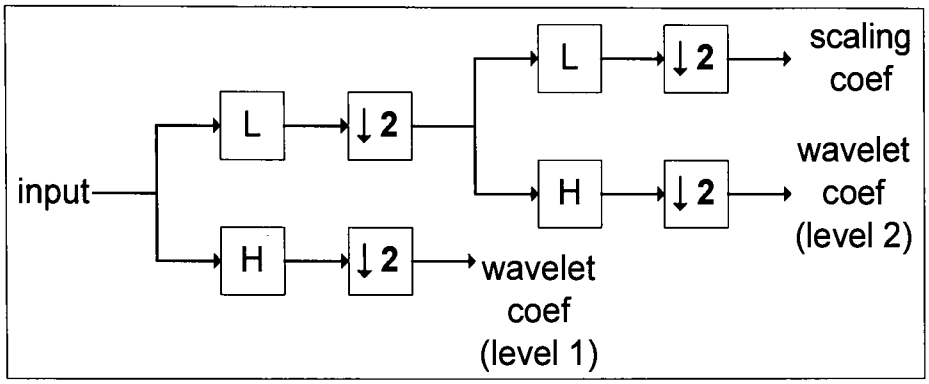
representation of the signal. Similarly in subband coding a time-scale representation of a digital signal is obtained using digital filtering techniques. The signal is passed through a series of high-pass filters and low-pass filters with different cut-off frequencies to analyze the high-frequency and low-frequency components of a signal respectively at different scales. The resolution of the signal is changed by the filtering operations and the scale is changed by upsampling and downsampling operations. The techniques used in subband coding can be applied to the DWT. Two digital filter banks are used to perform low-pass and high-pass filtering on the original signal, effectively splitting it into two frequency spectrums, or subbands. Each subband is down-sampled by a factor of two to keep the total number of samples the same as the original signal. The samples in the low-pass subband are referred to as the "scale" (scaling) coefficients with the low-pass filter being the scaling filter. The scale coefficients are also commonly referred to as "average", "approximation", or "smooth" coefficients as the low-pass filtering serves to smooth the original signal. The samples in the high-pass subband are referred to as the "wavelet" coefficients with the high-pass filter being the wavelet filter. The wavelet coefficients are also referred to as "detail" or "difference" coefficients as the high-pass filtering serves to highlight regions of larger variance. These wavelet coefficients contain the smallest details of interest, however, more detail information is present in the new low-pass subband of the signal. The DWT analysis of a signal using the described two-channel subband technique is shown in Figure 2.8.

Figure 2.8: DWT Analysis of Signal using Two-Channel Subband Coding



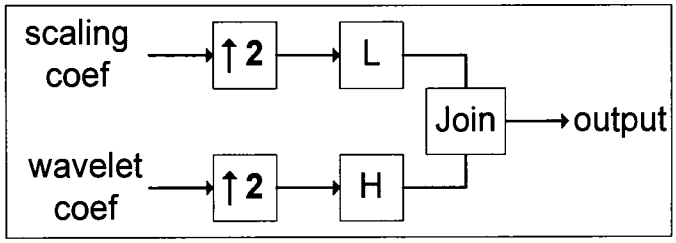
The procedure described above can be applied recursively to each resulting low-pass subband for multiple levels of decomposition, or analysis, of the signal as shown in Figure 2.9. In doing so, an iterated filter bank has been developed requiring only two filters, however, only providing fixed coverage of the signal spectrum.

Figure 2.9: Multiple Level DWT Analysis of Signal using Two-Channel Subband Coding



Reconstruction of the signal, or synthesis, is performed in the opposite manner by using synthesis filters and upsampling as demonstrated in Figure 2.10.

Figure 2.10: DWT Synthesis of Signal using Two-Channel Subband Coding



2.3. Biorthogonal Wavelets

A transform is described as being orthonormal if both its forward and inverse transforms are identical, therefore an orthonormal wavelet is one that is used in both analysis and synthesis of a signal. A filter having linear phase is one whose impulse response is either symmetric or anti-symmetric. Linear phase is important for a variety of reasons in applications where the signal is of finite duration, such as image compression. As mentioned earlier, two-channel subband transforms are used to perform the DWT on a signal. Unfortunately, there are no two-channel linear-phase subband filters with finite support that are also orthonormal. The solution is to use two symmetric wavelets for analysis and synthesis that are orthogonal to each other, or biorthogonal. These biorthogonal wavelets exhibit linear phase and therefore are now useful. [7 (chapter 4,6)] A family of biorthogonal wavelets that has proved useful in applications such as image compression is the Cohen-Daubechies-Feauveau (CDF) wavelet family. The CDF 5/3 and

CDF 9/7 are two specific wavelets that will be used as continuing examples throughout this thesis as they provide an interesting comparison.

2.3.1. CDF 5/3

The Cohen-Daubechies-Feauveau (CDF) 5/3 biorthogonal wavelet is a simple wavelet that has two sets of scaling and wavelet functions for analysis and synthesis, hence biorthogonality. The CDF 5/3 wavelet has a 5-tap low-pass analysis filter $h(z)$ and 3-tap high-pass analysis filter $g(z)$, hence 5/3. The CDF 5/3 also has a 3-tap low-pass synthesis filter $\tilde{h}(z)$ and 5-tap high-pass synthesis filter $\tilde{g}(z)$. The CDF 5/3 analysis and synthesis wavelets are shown in Figures 2.11 and 2.12 respectively.

Figure 2.11: CDF 5/3 Analysis Wavelet

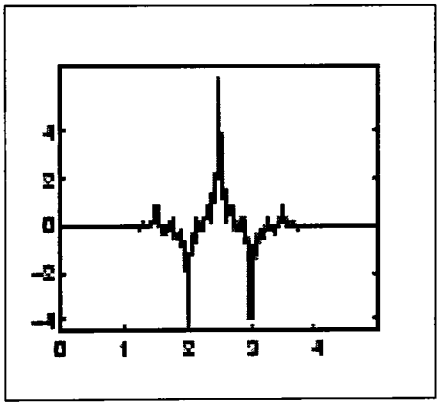
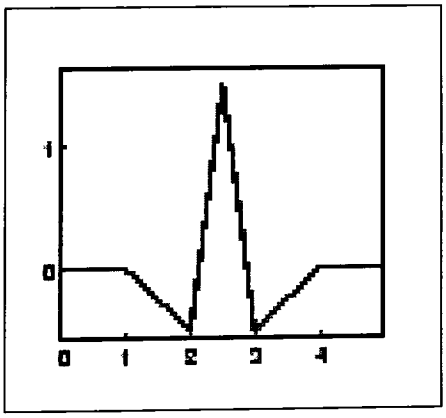


Figure 2.12: CDF 5/3 Synthesis Wavelet



The CDF 5/3 analysis and synthesis sequences are listed below.

Analysis Filters:

$$h(z) = -\frac{1}{8}z^{-2} + \frac{1}{4}z^{-1} + \frac{3}{4} + \frac{1}{4}z^{1} - \frac{1}{8}z^{2}$$

$$g(z) = -\frac{1}{2}z^{-1} + 1 - \frac{1}{2}z^{1}$$

Synthesis Filters:

$$\tilde{h}(z) = \frac{1}{2}z^{-1} + 1 + \frac{1}{2}z^{1}$$

$$\tilde{g}(z) = -\frac{1}{8}z^{-2} - \frac{1}{4}z^{-1} + \frac{3}{4} - \frac{1}{4}z^{1} - \frac{1}{8}z^{2}$$

2.3.2. CDF 9/7

The Cohen-Daubechies-Feauveau (CDF) 9/7 biorthogonal wavelet is a more complex wavelet than the CDF 5/3 wavelet. It also has two sets of scaling and wavelet functions for analysis and synthesis, however, they are nearly identical and therefore more orthonormal than the CDF 5/3. The CDF 9/7 wavelet has a 9-tap low-pass analysis filter $h(z)$ and 7-tap high-pass analysis filter $g(z)$. The CDF 9/7 also has a 7-tap low-pass synthesis filter $\tilde{h}(z)$ and 9-tap high-pass synthesis filter $\tilde{g}(z)$. The CDF 9/7 analysis and synthesis wavelets are shown in Figures 2.13 and 2.14 respectively.

Figure 2.13: CDF 9/7 Analysis Wavelet

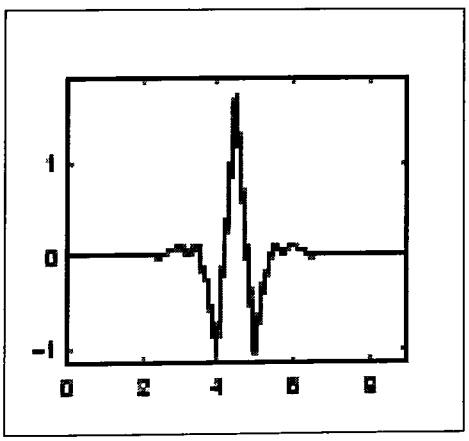
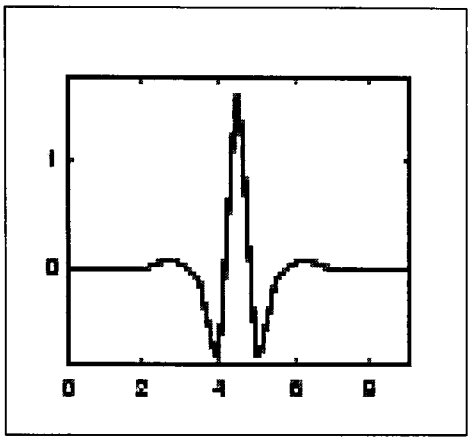


Figure 2.14: CDF 9/7 Synthesis Wavelet



The CDF 9/7 analysis and synthesis sequences are listed below.

Analysis Filters:

$$h(z) = \begin{pmatrix} 0.6029490182363579 \\ +0.2668641184428723\left(z^{-1}+z^{1}\right) \\ -0.07822326652898785\left(z^{-2}+z^{2}\right) \\ -0.01686411844287495\left(z^{-3}+z^{3}\right) \\ +0.02674875741080976\left(z^{-4}+z^{4}\right) \end{pmatrix}$$

$$g(z) = \begin{pmatrix} 1.115087052456994 \\ -0.5912717631142470\left(z^{-1}+z^{1}\right) \\ -0.05754352622849957\left(z^{-2}+z^{2}\right) \\ +0.09127176311424948\left(z^{-3}+z^{3}\right) \end{pmatrix}$$

Synthesis Filters:

$$\tilde{h}(z) = \begin{pmatrix} 1.115087052456994 \\ +0.5912717631142470\left(z^{-1}+z^{1}\right) \\ -0.05754352622849957\left(z^{-2}+z^{2}\right) \\ -0.09127176311424948\left(z^{-3}+z^{3}\right) \end{pmatrix}$$

$$\tilde{g}(z)=\begin{pmatrix} 0.6029490182363579 \\ -0.2668641184428723\left(z^{-1}+z^{1}\right) \\ -0.07822326652898785\left(z^{-2}+z^{2}\right) \\ +0.01686411844287495\left(z^{-3}+z^{3}\right) \\ +0.02674875741080976\left(z^{-4}+z^{4}\right) \end{pmatrix}$$

2.4. DWT using Convolution

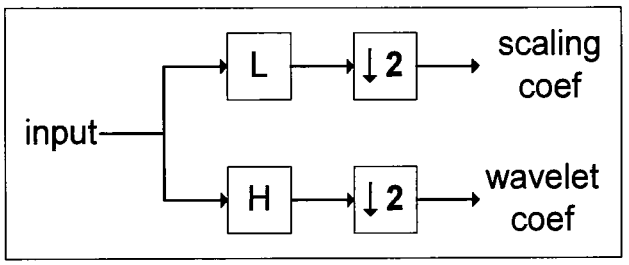
The traditional method for performing the DWT on a discrete signal is by convolution using digital filter banks. Convolution consists of performing a series of dot products between a set of filter coefficients and the signal. Convolution using digital filters is performed by inputting a sample along with a fixed number of its neighboring samples to the digital filter. A set of filter coefficients is used to evaluate the sample and its neighbors to get a new weighted, or convolved, value. The weighting operation is performed by multiplying all the samples by each of their respective filter coefficient values. The resulting products are then added together to get the final convolved value. The digital filter is shifted along the signal to repeat the above operation for every sample in the original signal and the final result is the new convolved signal.

One issue encountered during convolution arises when sufficient neighboring samples do not exist for a sample, as is the case for samples near the boundaries of a finite signal. An extension method needs to be implemented to provide the missing samples to the digital filter. A boundary extension method is then required to determine how to extend the signal beyond its boundaries to provide the extra samples needed by the filter. One method is to simply make all the extra values outside the boundary of the signal one constant value. An example of this method is to simply extend the sample at the edge of the signal beyond the boundary so that any neighbor sample outside the boundary is considered to be the boundary sample. Another boundary extension method is called symmetric extension. In symmetric extension the signal is reflected across the boundary providing a mirror image of the original signal as the extra samples. Whether the boundary sample is repeated or not beyond the boundary is another consideration.

To perform the forward DWT on a signal the appropriate filter coefficients first need to be derived from the scaling and wavelet functions. Once these coefficients are obtained, two digital

filters are constructed directly, one for the low-pass scaling filter and one for the high-pass wavelet filter. Once these two digital filters are obtained, the forward DWT is performed exactly as mentioned earlier using the two-channel subband coding technique. Convolution is performed on the signal using both the scaling and wavelet digital filters. The appropriate extension method is used to provide the missing samples. The resulting signals are downsampled by a factor of two to keep the same number of samples as the original signal. In this manner the scaling and wavelet coefficients are obtained as shown in Figure 2.15. Reconstruction of the signal is performed in the opposite manner using synthesis filters.

Figure 2.15: Forward DWT using Convolution



The example below demonstrates the steps required to perform the forward DWT for one level of decomposition on a 1-D discrete signal using convolution. The CDF 5/3 DWT is used. Symmetric extension of the signal is used to provide samples to the convolution filters beyond the boundaries of the signal.

Row of samples:	4	7	3	5	9	6

Convolution is performed on the original row of samples using the CDF 5/3 high-pass wavelet analysis filter. The results from the convolution are rounded using the floor function and then downsampled by a factor 2 starting with the second sample to obtain the wavelet coefficients of the signal.

$$g(z) = -\frac{1}{2}z^{-1} + 1 - \frac{1}{2}z^{1}$$

Row of samples:	4	7	3	5	9	6
Convolution output:	-3	**3**	-3	**-1**	3	**-3**

Wavelet coefficients: 3 -1 -3

Convolution is also performed on the original row of samples using the CDF 5/3 low-pass scaling analysis filter. The results from the convolution are rounded using the floor function and then downsampled by a factor 2 starting with the first sample to get the scaling coefficients of the signal.

$$h(z) = -\frac{1}{8}z^{-2} + \frac{1}{4}z^{-1} + \frac{3}{4} + \frac{1}{4}z^{1} - \frac{1}{8}z^{2}$$

Row of samples:	4	7	3	5	9	6
Convolution output:	**5**	5	**3**	5	**8**	7

Scaling coefficients: 5 3 8

It is important to note that temporary storage is required to hold the results of the convolution as the original samples cannot be overwritten until both the wavelet and scaling coefficients are obtained. Performing the DWT via convolution is a relatively expensive operation requiring large amounts of intermediate storage and unnecessary computations. In addition a DWT performed via convolution is limited just as the Fourier Transform is near the boundaries of a finite signal. A proposed alternative is the Lifting Scheme.

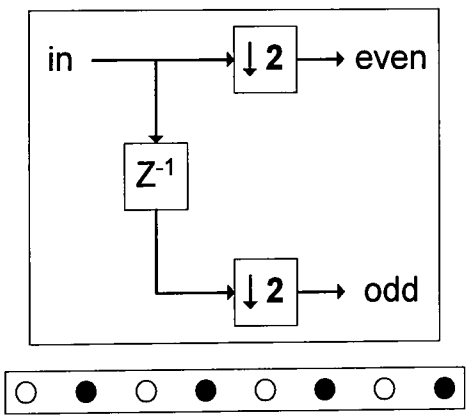
2.5. DWT using Lifting Scheme

In 1994 Wim Sweldens proposed an alternative approach to computing the DWT using biorthogonal wavelets called the Lifting Scheme. The lifting scheme calculates the DWT using spatial domain analysis rather than frequency domain analysis. This provides a more practical approach to computing the DWT that is better suited for many applications not requiring more complex frequency-based Fourier analysis techniques. In addition, certain information within the signal that cannot be obtained using traditional frequency-based techniques can be obtained using the lifting scheme and are referred to as second-generation wavelets. The lifting scheme can be traced back to the Euclidean algorithm and was inspired by M. Lounsbery's work concerning wavelet transform of meshes and D. L. Donoho's work concerning interpolation wavelet transforms, both of which are special cases of lifting. Lifting is also closely related to filter bank constructions and local decompositions. The lifting scheme attempts to provide a

more compact representation of a signal by finding and exploiting spatial correlation within. A major assumption that the lifting scheme takes advantage of is that samples that are spatially close to one another are more similar to each other than samples that are spatially farther away. Therefore neighboring samples are maximally correlated. Using this assumption a sample could be interpolated fairly accurately by having only knowledge of its close neighbors. A sample could be interpolated even more accurately the more neighbors there are to interpolate from. [8]

The lifting scheme consists of a series of steps that modify, or lift, one set of samples to be used in the next step. This "lifting" of samples gives the lifting scheme its name. The first step to performing the forward DWT using the lifting scheme is called the "Split" step shown in Figure 2.16. The purpose of the split step is to split the original signal into two sets that are maximally correlated. Given the assumption above neighboring samples are the most correlated so therefore during the split step the signal is split into one set containing only the even samples and a second set containing only the odd samples. This is commonly referred to as a Lazy wavelet transform.

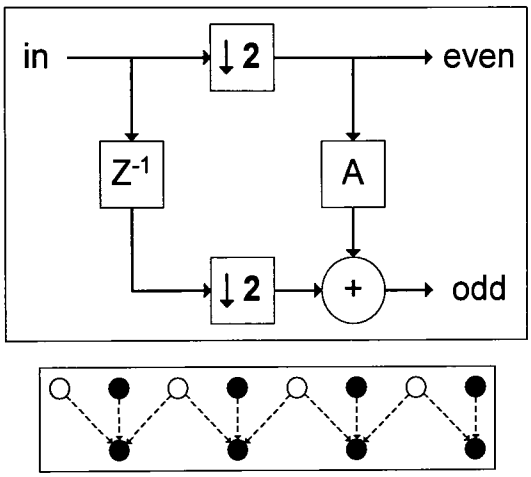
Figure 2.16: Split Step of Forward DWT using Lifting Scheme



The second step of the lifting scheme is called the "Predict" step shown in Figure 2.17. The predict step first uses samples from the even set to predict the values in the odd set. More specifically the right and left even neighbors of an odd sample are used in the prediction. Once the prediction is done the predicted value derived from the even set is compared to actual value from the odd set and the difference between the two is calculated. This difference should be relatively small compared to the original value from the odd set assuming the prediction method is appropriate. This difference is referred to as the "difference" or "wavelet" coefficient. The

wavelet coefficient indicates the extent to which the prediction of the odd set of samples from the even set of samples fail. If the wavelet coefficient is zero this means the sample from the odd set can be exactly predicted from the samples in the even set. Finally each sample in the odd set is overwritten with their respective wavelet coefficients, again which should be relatively small. The odd set now contains the wavelet coefficients and the even set still contains the even samples from the original signal. The odd set captures the high-frequency content of the original signal now, but the even set still contains both high and low-frequency data from the original signal since it is merely a downsampled version and thus contains aliasing. A third step is needed to remove the aliasing.

Figure 2.17: Split and Predict Steps of Forward DWT using Lifting Scheme

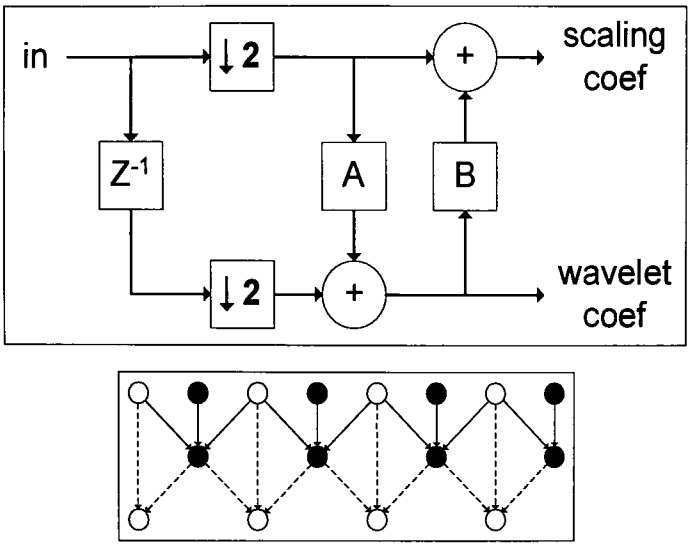


The following is the predict step equation with α as the predict step coefficient.

Predict: $$odd_{new} = odd_{old} + \alpha(even_{left} + even_{right})$$

The third step of the lifting scheme is called the "Update" step shown in Figure 2.18. The Update step uses the new wavelet coefficients in the odd set to update the even set producing "smooth" or "scaling" coefficients. More specifically the right and left odd neighbors of an even sample are used in the update. These scaling coefficients now have same average intensity of the original signal in the form of low-frequency content and thus remove the aliasing. The scaling coefficients now overwrite their respective samples in the even set. The signal now contains the wavelet and scaling coefficients and thus the forward DWT has been performed.

Figure 2.18: Split, Predict, and Update Steps of Forward DWT using Lifting Scheme



The following is the update step equation with β as the update step coefficient.

Update: $$even_{new} = even_{old} + \beta(odd_{left} + odd_{right})$$

A more complex DWT involving more neighbors can be performed by merely adding more predict and update steps to the method described above, with the final predict and update steps producing the wavelet and scaling coefficients respectively as shown in Figure 2.19.

In addition, scaling steps can be added to adjust the DC gain of the wavelet and scaling coefficients if needed as shown in Figure 2.20.

The inverse DWT using the lifting scheme is intuitive. To perform the inverse DWT the lifting steps are simply performed in the reverse order using the inverse operations. If any scaling was performed on the wavelet and scaling coefficients at the end of the forward DWT then it first must be undone by scaling the coefficients by inverse scale factors. Next the last update step must be undone followed by the last predict step. This continues backwards until all the steps have been undone and the original signal has been obtained as demonstrated in Figures 2.21 and 2.22.

Figure 2.19: Forward DWT using Lifting Scheme with Multiple Lifting Steps

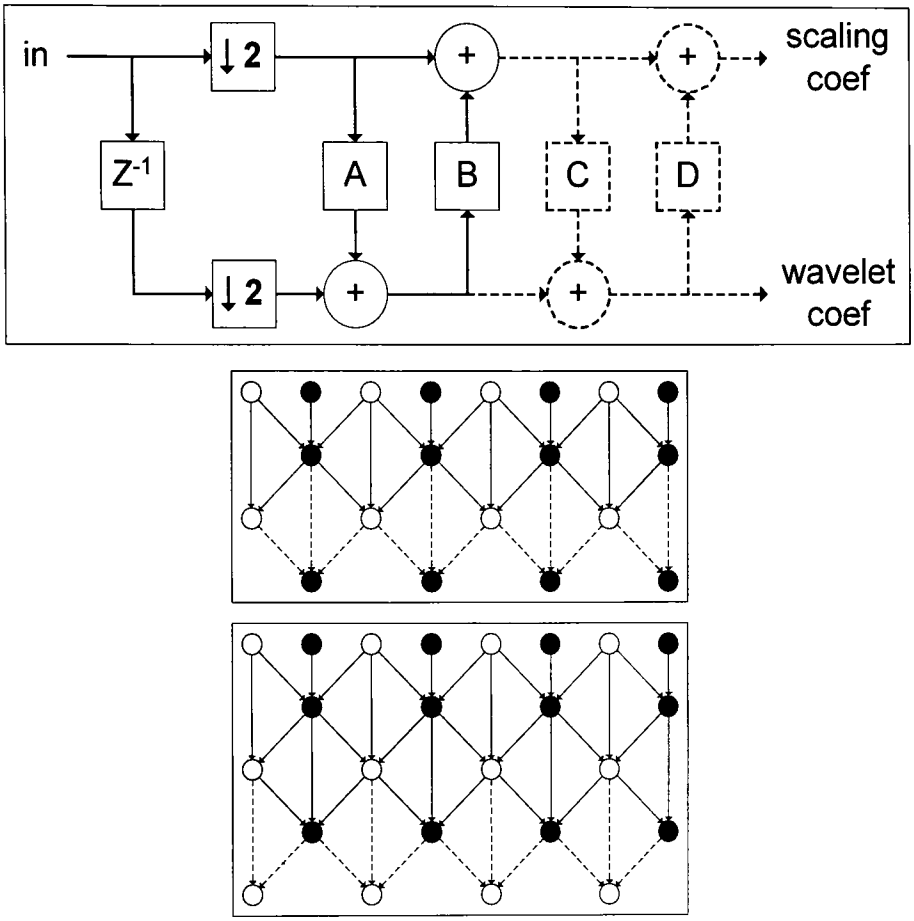
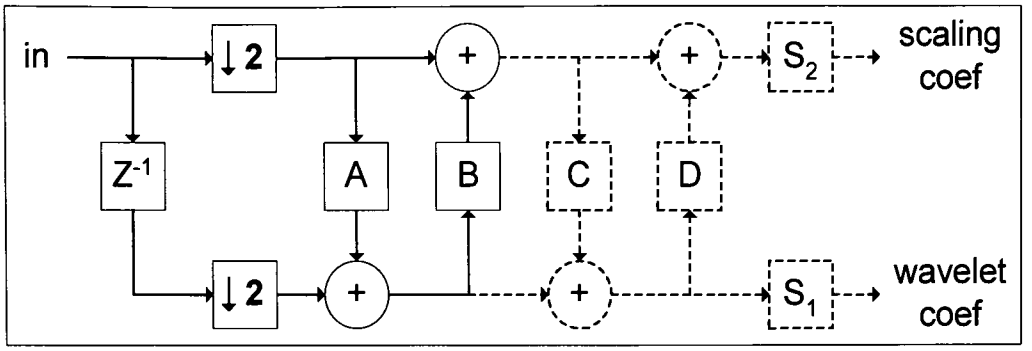


Figure 2.20: Forward DWT using Lifting Scheme with Multiple Lifting and Scaling Steps



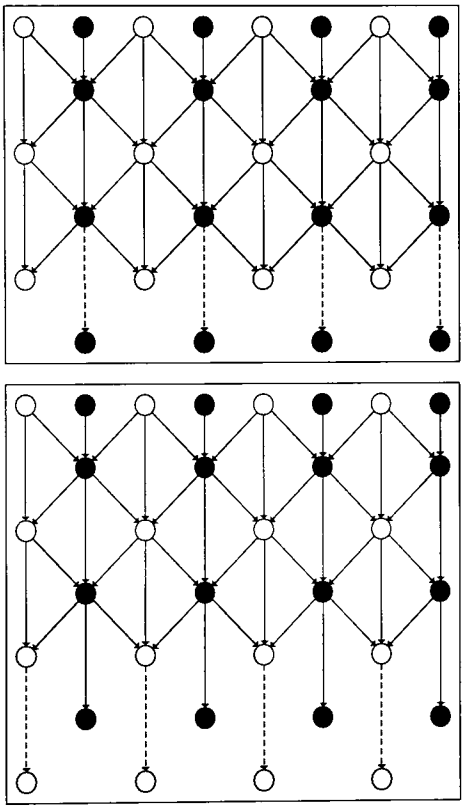


Figure 2.21: Inverse DWT via Lifting Scheme

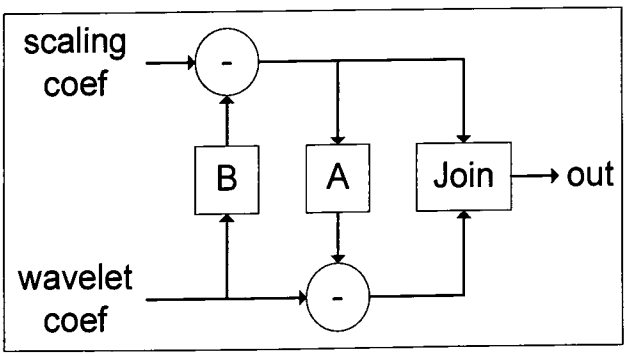
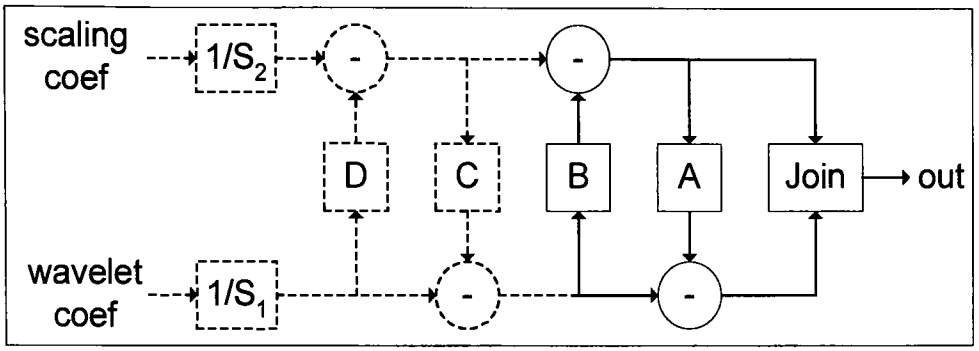


Figure 2.22: Inverse DWT using Lifting Scheme with Multiple Lifting and Scaling Steps



2.5.1. Features of Lifting Scheme

The main advantages of computing the DWT via the lifting scheme rather than convolution are that it is much faster, is calculated fully in-place requiring no intermediate storage, has symmetric forward and inverse transforms, potential for integer wavelet transform, and can be used in situations where frequency-based techniques cannot. [8]

2.5.1.1. Faster Implementation

The traditional DWT is performed using a two-band subband analysis whereby both a high-pass and a low-pass analysis filter are applied to the entire signal and the resulting high-pass and low-pass subbands are downsampled by a factor of two to get the result. The lifting scheme exploits the similarities between the two analysis filters to reduce the number of calculations. Unnecessary calculations resulting in samples that would be lost in the downsampling and repeat calculations between the two analysis filters are avoided thus providing an overall speedup. In some cases the number of operations can be reduced by up to a factor of two. [9]

2.5.1.2. In-Place Calculation

The lifting scheme computes the DWT in an iterative manner. After each iteration the intermediate results/samples can overwrite the previous results/samples without data loss and therefore no auxiliary memory is required in the calculation of the DWT. The lifting scheme is therefore said to provide a fully in-place calculation of the DWT. [9]

2.5.1.3. Symmetric Forward and Inverse Transform

Traditionally the inverse DWT is performed by using a set of synthesis filters separate from the analysis filters and as a result the inverse transform may not be intuitive. The lifting scheme performs the inverse DWT by simply undoing the operations of the forward DWT. The same machinery used for the forward DWT can be used for the inverse DWT by reversing the order of operations.

2.5.1.4. Integer-to-Integer Transform

Due to the linearity of the lifting scheme, if the input data is in integer format, it is possible to maintain data to be in integer format throughout the transform by introduction a rounding function in the filtering operation. Due to this property, the transform is reversible and is called the integer-to-integer, or Integer Wavelet Transform (IWT).

2.5.1.5. Second Generation Wavelets

Since the lifting scheme does not use Fourier analysis to compute the DWT, it can be used in situations where translation and dilation is impossible. One example would be near boundaries of a finite signal where normal Fourier techniques would provide border distortion or artifacts.

2.5.2. Lifting Step Extraction

As mentioned above the lifting scheme is an alternative technique for performing the DWT using biorthogonal wavelets. In order to perform the DWT using the lifting scheme the corresponding lifting and scaling steps must be derived from the biorthogonal wavelets. The analysis filters of the particular wavelet are first written in polyphase matrix form shown below.

$$P(z)=\begin{bmatrix} h_{even}(z) & g_{even}(z) \\ h_{odd}(z) & g_{odd}(z) \end{bmatrix}$$

The polyphase matrix is a 2 x 2 matrix containing the analysis low-pass and high-pass filters each split up into their even and odd polynomial coefficients and normalized. From here the matrix is factored into a series of 2 x 2 upper and lower triangular matrices each with diagonal entries equal to 1. The upper triangular matrices contain the coefficients for the predict steps and the lower triangular matrices contain the coefficients for the update steps. A matrix consisting of all 0's with the exception of the diagonal values may be extracted to derive the scaling step

coefficients. The polyphase matrix is factored into the form shown in the equation below, α is the coefficient for the predict step and β is the coefficient for the update step.

$$P(z)=\begin{bmatrix}1 & \alpha(1+z^{-1})\\ 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0\\ \beta(1+z) & 1\end{bmatrix}$$

An example of a more complicated extraction having multiple predict and update steps as well as scaling steps is shown below; α is the coefficient for the first predict step, β is the coefficient for the first update step, λ is the coefficient for the second predict step, δ is the coefficient for the second update step, ζ_1 is the odd sample scaling coefficient, and ζ_2 is the even sample scaling coefficient.

$$P(z)=\begin{bmatrix}1 & \alpha(1+z^{-1})\\ 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0\\ \beta(1+z) & 1\end{bmatrix}\begin{bmatrix}1 & \gamma(1+z^{-1})\\ 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0\\ \delta(1+z) & 1\end{bmatrix}\begin{bmatrix}\zeta_1 & 0\\ 0 & \zeta_2\end{bmatrix}$$

According to matrix theory, any matrix having polynomial entries and a determinant of 1 can be factored as described above. Therefore every FIR wavelet or filter bank can be decomposed into a series of lifting and scaling steps. Daubechies and Sweldens discuss lifting step extraction in further detail. [10] The lifting step extraction for the CDF 5/3 and CDF 9/7 biorthogonal wavelets is shown below.

2.5.3. CDF 5/3 DWT using Lifting Scheme

The low-pass and high-pass analysis filters for the CDF 5/3 are restated below with the high-pass filter translated by z^{-1}.

$$h(z)=-\frac{1}{8}z^{-2}+\frac{1}{4}z^{-1}+\frac{3}{4}+\frac{1}{4}z^{1}-\frac{1}{8}z^{2}$$

$$z^{-1}g(z)=z^{-1}\left(-\frac{1}{2}z^{-1}+1-\frac{1}{2}z^{1}\right)=-\frac{1}{2}z^{-2}+z^{-1}-\frac{1}{2}$$

The polyphase matrix $P(z)$ for the CDF 5/3 wavelet is shown below.

$$P(z)=\begin{bmatrix}-\frac{1}{8}z^{-1}+\frac{3}{4}-\frac{1}{8}z & -\frac{1}{2}z^{-1}-\frac{1}{2}\\ \frac{1}{4}+\frac{1}{4}z & 1\end{bmatrix}$$

$$P(z)=\begin{bmatrix} \frac{3}{4}-\frac{1}{8}\left(z+z^{-1}\right) & -\frac{1}{2}\left(1+z^{-1}\right) \\ \frac{1}{4}(1+z) & 1 \end{bmatrix}$$

The polyphase matrix can then be factored into two triangular matrices.

$$P(z)=\begin{bmatrix} 1 & -\frac{1}{2}\left(1+z^{-1}\right) \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 \\ \frac{1}{4}(1+z) & 1 \end{bmatrix}$$

It is apparent that two lift steps are required, one predict and one update step, to perform the CDF 5/3 DWT. The coefficient for the predict step is:

$$\alpha=-\frac{1}{2}$$

and the coefficient for the update step is:

$$\beta=\frac{1}{4}$$

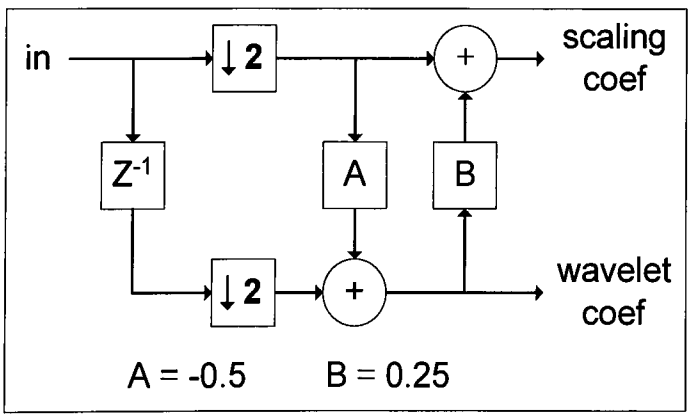
The predict and update equations for the CDF 5/3 filter are shown below.

Predict: $$\text{odd}_{new}=\text{odd}_{old}+\left\lfloor -\frac{1}{2}\left(\text{even}_{left}+\text{even}_{right}\right)\right\rfloor$$

Update: $$\text{even}_{new}=\text{even}_{old}+\left\lfloor \frac{1}{4}\left(\text{odd}_{left}+\text{odd}_{right}\right)\right\rfloor$$

The floor function is used for both the predict and update equations to provide an integer-to-integer transform. The forward CDF 5/3 DWT using the lifting scheme is shown in Figure 2.23.

Figure 2.23: Forward CDF 5/3 DWT using Lifting Scheme



The example below demonstrates the steps required to perform the forward DWT for one level of decomposition on a 1-D signal using the lifting scheme. The CDF 5/3 DWT is used. Symmetric extension of only one sample beyond the boundary is necessary for the lifting scheme.

Row of samples:	4	7	3	5	9	6

The CDF 5/3 DWT consists of two lifting steps. The first lifting step (predict step) is applied to the original row of samples and the results then safely overwrite the odd samples in the original signal for use in the next lifting step. The results from this step are the wavelet coefficients.

$$\text{odd}_{new} = \text{odd}_{old} + \left\lfloor -\frac{1}{2}\left(\text{even}_{left} + \text{even}_{right}\right) \right\rfloor$$

Row of samples:	4	7	3	5	9	6
Lifting step 1 results:		3		-1		-3
New row of samples:	4	**3**	3	**-1**	9	**-3**
Wavelet coefficients:	**3**		**-1**		**-3**	

The second lifting step (update step) is applied to the new wavelet coefficients and the remaining even samples of the original signal. The results then safely overwrite the even samples in the signal. The results from this step are the scaling coefficients.

$$\text{even}_{new} = \text{even}_{old} + \left\lfloor \frac{1}{4}\left(\text{odd}_{left} + \text{odd}_{right}\right) \right\rfloor$$

Row of samples:	4	3	3	-1	9	-3
Lifting step 2 results:	5		3		8	
New row of samples:	**5**	3	**3**	-1	**8**	-3
Scaling coefficients:	**5**		**3**		**8**	

The resulting wavelet and scaling coefficients obtained from this example are identical to those from the forward DWT example using convolution earlier.

The inverse DWT transform is performed by performing the inverted versions of the lifting steps on the wavelet and scaling coefficients in the reverse order as they were performed for the forward DWT. The wavelet coefficients are located in the odd sample positions and the scaling coefficients are located in the even sample positions.

Wavelet/Scaling Coefficients:	5	3	3	-1	8	-3

The inverse version of the second lifting step (update step) is applied to the wavelet and scaling coefficients and the results then safely overwrite the scaling coefficients. The results from this step are the even samples of the original signal.

$$\text{even}_{new} = \text{even}_{old} - \left\lfloor \frac{1}{4}\left(\text{odd}_{left} + \text{odd}_{right}\right) \right\rfloor$$

Wavelet/Scaling Coefficients:	5	3	3	-1	8	-3
Inverse lifting step 2						
Results:	4		3		9	
New row of samples:	**4**	3	**3**	-1	**9**	-3

The inverse version of the first lifting step (predict step) is applied to newly determined even samples and remaining wavelet coefficients. The results then safely overwrite the wavelet coefficients. The results from this step are the odd samples of the original signal.

$$\text{odd}_{new} = \text{odd}_{old} - \left\lfloor -\frac{1}{2}\left(\text{even}_{left} + \text{even}_{right}\right) \right\rfloor$$

New row of samples:	4	3	3	-1	9	-3
Inverse lifting step 1						
Results:		7		5		6

New row of samples: 4 **7** 3 **5** 9 **6**

The complete signal has been perfectly reconstructed.

Reconstructed signal: **4** **7** **3** **5** **9** **6**

2.5.4. CDF 9/7 DWT using Lifting Scheme

The low-pass and high-pass analysis filters for the CDF 9/7 are restated below with the high-pass filter translated by z^{-1}.

$$h(z)=\begin{pmatrix} 0.6029490182363579 \\ +0.2668641184428723\left(z^{-1}+z^{1}\right) \\ -0.07822326652898785\left(z^{-2}+z^{2}\right) \\ -0.01686411844287495\left(z^{-3}+z^{3}\right) \\ +0.02674875741080976\left(z^{-4}+z^{4}\right) \end{pmatrix}$$

$$z^{-1}g(z)=z^{-1}\begin{pmatrix} 1.115087052456994 \\ -0.5912717631142470\left(z^{-1}+z^{1}\right) \\ -0.05754352622849957\left(z^{-2}+z^{2}\right) \\ +0.09127176311424948\left(z^{-3}+z^{3}\right) \end{pmatrix}=\begin{pmatrix} 1.115087052456994z^{-1} \\ -0.5912717631142470\left(z^{-2}+1^{1}\right) \\ -0.05754352622849957\left(z^{-3}+z^{1}\right) \\ +0.09127176311424948\left(z^{-4}+z^{2}\right) \end{pmatrix}$$

The polyphase matrix $P(z)$ is trivial yet very ungainly and therefore not shown. The polyphase matrix for the CDF 9/7 wavelet can be factored into the following four triangular matrices and one scale matrix.

$$P(z)=\begin{bmatrix} 1 & \alpha\left(1+z^{-1}\right) \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 \\ \beta(1+z) & 1 \end{bmatrix}\begin{bmatrix} 1 & \gamma\left(1+z^{-1}\right) \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 \\ \delta(1+z) & 1 \end{bmatrix}\begin{bmatrix} \zeta_1 & 0 \\ 0 & \zeta_2 \end{bmatrix}$$

where:

$\alpha = -1.586134342$

$\beta = -0.05298011854$

$\gamma = 0.8829110762$

$\delta = 0.4435068522$

$\zeta_1 = 1.149604398$

$\zeta_2 = 0.8698644523$

A total of four lift steps are required, two predict and two update steps, to perform the CDF 9/7 DWT. The coefficient for the first predict step is:

$$\alpha = -1.586134342$$

The coefficient for the first update step is:

$$\beta = -0.05298011854$$

The coefficient for the second predict step is:

$$\gamma = 0.8829110762$$

The coefficient for the second update step is:

$$\delta = 0.4435068522$$

The scale coefficient for the odd samples is:

$$\zeta_1 = 1.149604398$$

The scale coefficient for the even samples is:

$$\zeta_2 = 0.8698644523$$

The predict and update equations for the CDF 9/7 filter are shown below.

Predict1: $\text{odd}_{new} = \text{odd}_{old} + \lfloor \alpha(\text{even}_{left} + \text{even}_{right}) \rfloor$

Update1: $\text{even}_{new} = \text{even}_{old} + \lfloor \beta(\text{odd}_{left} + \text{odd}_{right}) \rfloor$

Predict2: $\text{odd}_{new} = \text{odd}_{old} + \lfloor \gamma(\text{even}_{left} + \text{even}_{right}) \rfloor$

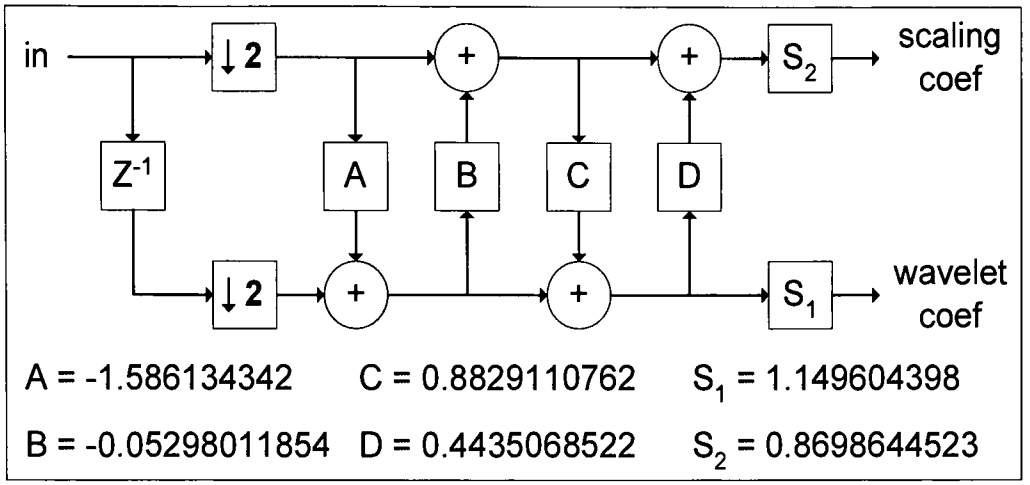
Update2: $\text{even}_{new} = \text{even}_{old} + \lfloor \delta(\text{odd}_{left} + \text{odd}_{right}) \rfloor$

Scale odd: $\text{odd}_{new} = \lfloor \zeta_1 \times \text{odd}_{old} \rfloor$

Scale even: $\text{even}_{new} = \lfloor \zeta_2 \times \text{even}_{old} \rfloor$

The floor function is used for all the predict, update and scale equations to provide an integer-to-integer transform. The forward CDF 9/7 DWT using the lifting scheme is shown in Figure 2.24.

Figure 2.24: Forward CDF 9/7 DWT using Lifting Scheme



The CDF 9/7 DWT consists of four lifting steps and two scaling steps. The first lifting step (predict step 1) is applied to the original row of samples and the results then safely overwrite the odd samples in the original signal for use in the next lifting step.

$$odd_{new} = odd_{old} + \lfloor \alpha(even_{left} + even_{right}) \rfloor$$

$$\alpha = -1.586134342$$

Row of samples:	4	7	3	5	9	6
Lifting step 1 results:		-5		-15		-23
New row of samples:	4	**5**	3	**-15**	9	**-23**

The second lifting step (update step 1) is applied to the results from the first lifting step and the remaining even samples of the original signal. The results then safely overwrite the even samples in the signal.

$$even_{new} = even_{old} + \lfloor \beta(odd_{left} + odd_{right}) \rfloor$$

$$\beta = -0.05298011854$$

Row of samples:	4	5	3	-15	9	-23
Lifting step 2 results:	4		4		11	
New row of samples:	**4**	5	**4**	-15	**11**	-23

The third lifting step (predict step 2) is applied to the results from the first and second lifting steps. The results then safely overwrite the results from the first lifting step.

$$\text{odd}_{\text{new}} = \text{odd}_{\text{old}} + \lfloor \gamma(\text{even}_{\text{left}} + \text{even}_{\text{right}}) \rfloor$$

$$\gamma = 0.8829110762$$

Row of samples:	4	5	4	-15	11	-23
Lifting step 3 results:		2		-2		-4
New row of samples:	4	**2**	4	**-2**	11	**-4**

The fourth lifting step (update step 2) is applied to the results from the second and third lifting steps. The results then safely overwrite the results from the second lifting step.

$$\text{even}_{\text{new}} = \text{even}_{\text{old}} + \lfloor \delta(\text{odd}_{\text{left}} + \text{odd}_{\text{right}}) \rfloor$$

$$\delta = 0.4435068522$$

Row of samples:	4	2	4	-2	11	-4
Lifting step 4 results:	5		4		8	
New row of samples:	**5**	2	**4**	-2	**8**	-4

The first scaling step is applied to the results from the third lifting step. The results then safely overwrite the results from the third lifting step. The results from this step are the wavelet coefficients.

$$\text{odd}_{\text{new}} = \lfloor \zeta_1 \times \text{odd}_{\text{old}} \rfloor$$

$$\zeta_1 = 1.149604398$$

Row of samples:	5	2	4	-2	8	-4
Lifting step 4 results:		2		-3		-5
New row of samples:	5	**2**	4	**-3**	8	**-5**

Wavelet coefficients:	**2**	**-3**	**-5**

The second scaling step is applied to the results from the fourth lifting step. The results then safely overwrite the results from the fourth lifting step. The results from this step are the scaling coefficients.

$$\text{even}_{new} = \lfloor \zeta_2 \times \text{even}_{old} \rfloor$$

$$\zeta_2 = 0.8698644523$$

Row of samples:	5	2	4	-3	8	-5
Lifting step 4 results:	4		3		6	
New row of samples:	4	2	**3**	-3	**6**	-5
Scaling coefficients:	4	**3**	**6**			

2.6. 2-D DWT and Digital Images

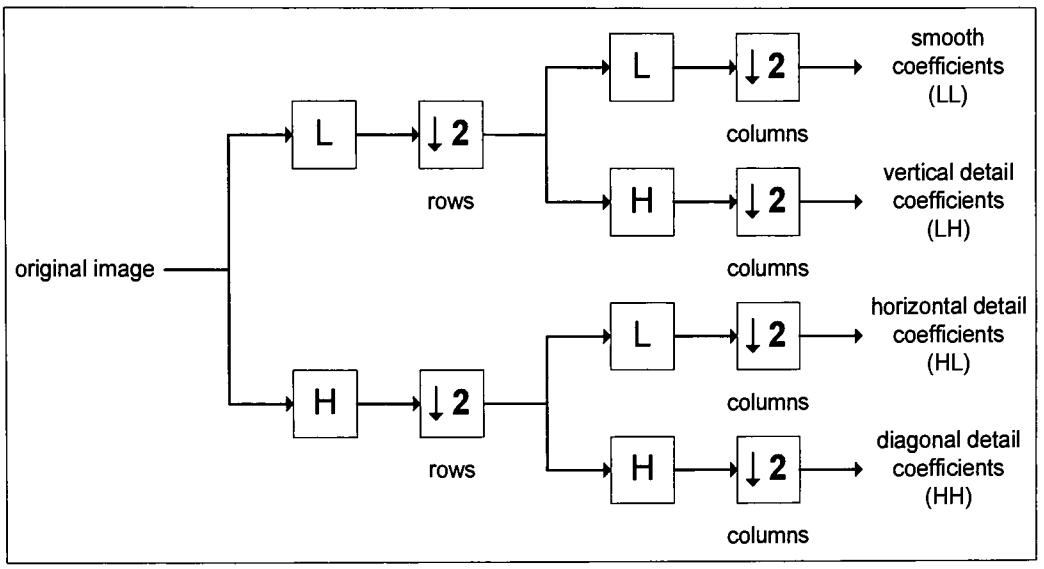
As still and motion pictures are steadily moving from the traditional film media into digital media form the demand for digital image processing techniques is growing. Since digital images are merely 2-dimensional spatial-domain signals, the field of digital image processing is simply a classification of signal processing pertaining specifically to digital images and thus many techniques that are used in signal processing are also applicable to digital images. Hence, the application of the DWT to digital images is of growing interest.

2.6.1. 2-D DWT

A digital image is an example of a spatial-domain signal having an amplitude, or intensity that varies depending on the location within the image. An image consists of picture elements, or "pixels", that represent intensity values at a specific location with the image. These pixels are located within a discrete 2-D grid of rows and columns that represent the spatial-domain. A grayscale image contains pixels having only one intensity value ranging from black to white creating a monochrome "black & white" image, also called an intensity image. A color image usually has three intensity values per pixel representing red, blue and green to create a full-color image. For the most part, image processing techniques that are unrelated to color are the same for both types of pictures and therefore for the purposes of this thesis only grayscale images will be used.

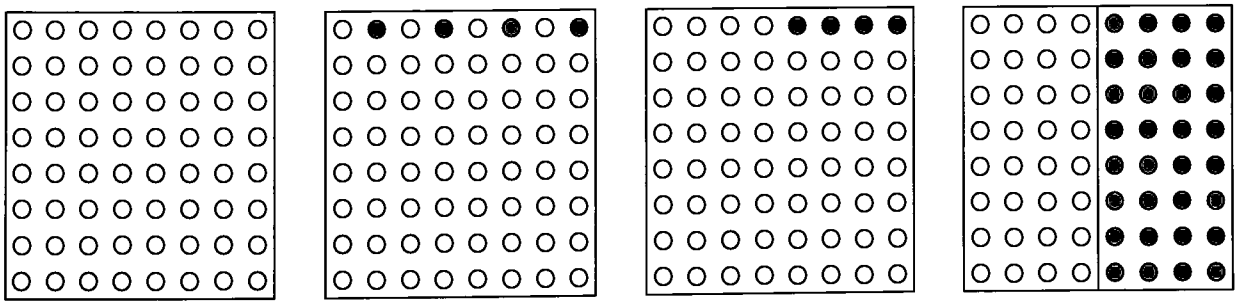
The DWT described earlier applies to 1-D signals, but can be easily modified to accompany 2-D signals such as digital images by treating them as groups of 1-D signals as shown in Figure 2.25.

Figure 2.25: Forward 2-D DWT



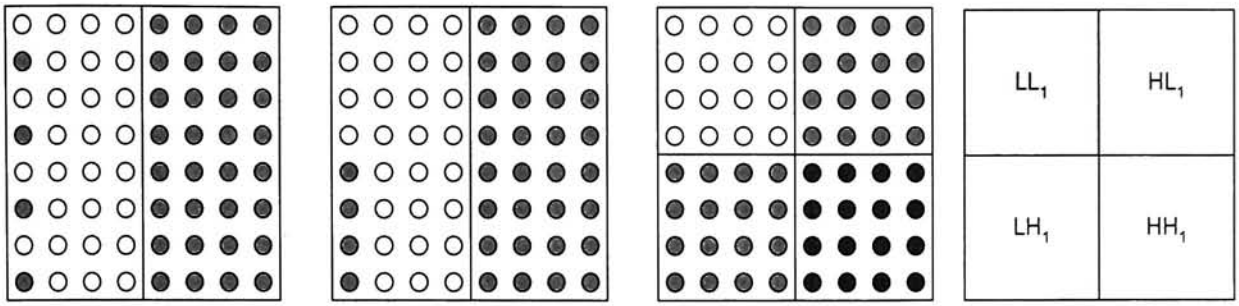
In order to perform the 2-D DWT on image the 1-D DWT is performed horizontally along each row (or vertically along the columns if desired) of the image treating each row as if they were single 1-D signals as shown in Figure 2.26. The results are two subbands, one containing the low-pass scaling coefficients and the other containing the high-pass wavelet coefficients.

Figure 2.26: Forward 2-D DWT Row Processing of Image



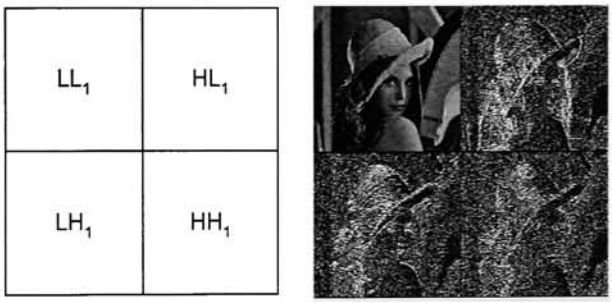
After the 1-D DWT is performed on all the rows of the image, the process is repeated vertically on every column of the two subbands from the previous step as shown in Figure 2.27. The two original subbands are each split into two more subbands for a total of four subbands. Once this is complete one level of decomposition has been performed on the image.

Figure 2.27: Forward 2-D DWT Column Processing of Image



The LL subband contains the scaling coefficients resulting from the 2-D DWT of the image which visually appears to be a compressed "smoothed" version of the original image. The HL, LH, and HH subbands contain the horizontal, vertical, and diagonal wavelet coefficients, or "details", of the image respectively. The details are regions of sharp change in the image. An example of a transformed image is shown in Figure 2.28.

Figure 2.28: Forward 2-D DWT of Image



A multiple-level decomposition can be performed by recursively applying the 2-D DWT to the LL subband as shown in Figure 2.29.

Figure 2.30 shows the row and column processing for the second level of decomposition. An example of a transformed image for two levels of decomposition is shown in Figure 2.31. This nested four-subband representation, or dyadic decomposition, of a transformed image is commonly referred to as a "quad-tree" .

Figure 2.29: Forward 2-D DWT for Multiple Levels of Decomposition

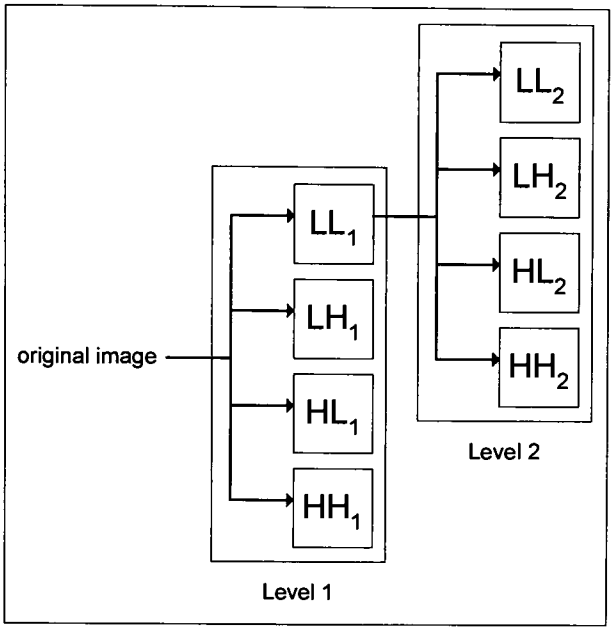


Figure 2.30: Forward 2-D DWT Row and Column Processing of Image for Multiple Levels of Decomposition

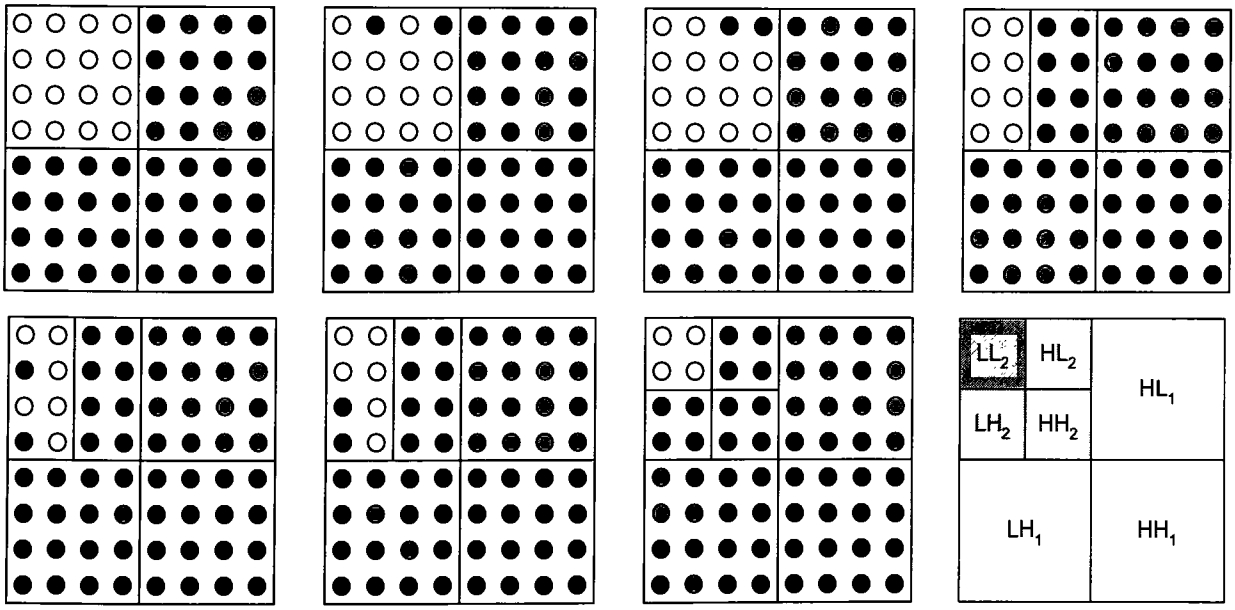
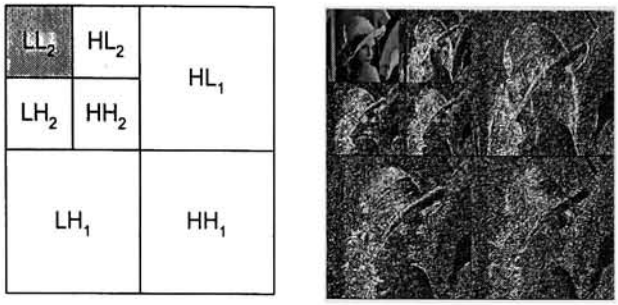


Figure 2.31: Forward 2-D DWT of Image for Multiple Levels of Decomposition



2.6.1.1. Finite precision effects:

The scaling and wavelet coefficients resulting from 2-D integer-to-integer DWT require slightly larger precision to avoid the risk of numerical overflow or underflow. Tables 2.1 and 2.2 from [7 (chapter 17)] show the fractional number of extra bits necessary to represent the coefficients in each of the four subbands of a transformed image for 1-5 and 15 levels of decomposition using both the CDF 5/3 and CDF 9/7 DWT and without causing overflow or underflow.

For the CDF 5/3 the number of extra bits required to eliminate overflow or underflow increases slightly the more levels of decomposition that are performed. Also the number of extra bits required is smallest for the LL subband and the largest for the HH subband, with the LH/HL subbands in between. From Table 2.1 it is determined that four extra guard bits for the wavelet and scaling coefficients would prevent overflow or underflow when performing the CDF 5/3 DWT.

In contrast to the CDF 5/3, for the CDF 9/7 the number of extra bits required to eliminate overflow or underflow decreases slightly the more levels of decomposition that are performed. Also the number of extra bits required is smallest for the HH subband and the largest for the LL subband, with the LH/HL subbands in between. From Table 2.2 it is determined that an one extra guard bit for the wavelet and scaling coefficients would prevent overflow or underflow when performing the CDF 9/7 DWT.

Table 2.1: Number of Guard Bits Needed for CDF 5/3 DWT Scaling and Wavelet Coefficients to Eliminate Overflow and Underflow

# Levels of Decomposition	Subband		
	LL	LH, HL	HH
1	1.170	1.585	2.000
2	1.401	2.022	2.644
3	1.510	2.214	2.919
4	1.525	2.250	2.976
5	1.542	2.267	2.991
15	1.557	2.298	3.039

Table 2.2: Number of Guard Bits Needed for CDF 9/7 DWT Scaling and Wavelet Coefficients to Eliminate Overflow and Underflow

# Levels of Decomposition	Subband		
	LL	LH, HL	HH
1	0.930	0.841	0.752
2	0.829	0.807	0.785
3	0.772	0.737	0.703
4	0.763	0.686	0.609
5	0.757	0.679	0.600
15	0.755	0.674	0.592

2.6.2. 2-D DWT Properties and Digital Images

The DWT has many properties that make it an attractive digital image processing tool suitable to many applications. The primary properties of wavelet transforms make wavelet-domain statistical image processing attractive: multiresolution, locality, decorrelation, edge detection, and energy compaction. The secondary properties of wavelet transforms based off of the primary properties are: NonGaussianity and Persistency. The tertiary properties of wavelet transforms are exponential decay across scale and stronger persistence at finer scales. The features of each are defined below. [11]

2.6.2.1. Multiresolution

As mentioned earlier the DWT provides multiresolution analysis. The quad-tree representation of an image represents the image at a nested set of scales. The scale produced from the first level of decomposition is said to be at the finest scale with each successive level of decomposition being at a coarser scale. Multiresolution is powerful because details within an image may be more readily apparent at differing scales.

2.6.2.2. Locality

Each wavelet represents the image content localized in spatial location and frequency.

2.6.2.3. Decorrelation

The wavelet coefficients of images tend to be approximately decorrelated.

2.6.2.4. Edge Detection

Wavelets act as local edge detectors by representing edges in the image by large wavelet coefficients at the corresponding spatial location.

2.6.2.5. Energy Compaction

The wavelet coefficients of images tend to be sparse. The wavelet coefficient is large only if edges are present within the support of the wavelet.

2.6.2.6. NonGaussianity

The wavelet coefficients of images have peaky, heavy-tailed marginal distributions.

2.6.2.7. Persistency

The wavelet coefficients of images have large and small values that tend to propagate through the scales of the quad-trees.

2.6.2.8. Exponential Decay Across Scales

The magnitudes of wavelet coefficients of images tend to decay exponentially across scales.

2.6.2.9. Stronger Persistence at Finer Scales

The persistence of large and small wavelet coefficient magnitudes becomes stronger at finer scales.

2.7. Wavelet Applications to Digital Images

DWT has been proven useful in such areas as image compression, de-noising, edge-detection, feature extraction, and image classification to name a few.

2.7.1. Image Denoising

Image denoising involves removing unwanted artifacts, or noise, within an image. The goal is to produce a clearer image that is more suitable for its intended application. Noise is introduced into digital images primarily during image acquisition and transmission. Unfavorable environmental conditions and poor quality of sensing elements can introduce noise into an image during image acquisition. Noise from outside sources or the transmission line may interfere with an image during transmission. Some noise probability density function (PDF) models include Gaussian and impulse (salt-and-pepper, speckle) noise. Gaussian noise in an image can be due to factors such as electronic circuit and sensor noise due to poor illumination and/or high temperature. Impulse noise is found in situation where quick transients, such as faulty switching take place during imaging. [4 (chapter 5)]

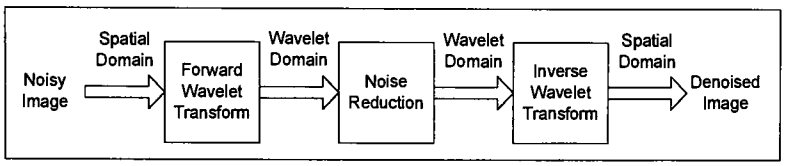
Many proven methods exist to denoise images such as spatial-domain adaptive mean and median filters to frequency-domain pass filters. While some filters are better at denoising specific types of noise than others, none are ideally suited for all types of noise. Also many denoising techniques require insight into the noise PDF before denoising takes place.

2.7.1.1. Denoising using Wavelets

One of the popular applications of the DWT is in the area of signal denoising. The DWT has proven to be a very simple and effective tool in image denoising. The one major advantage of wavelet denoising is that in many cases no priori information about the noise probability density function is necessary to successfully denoise an image [12]. Wavelet denoising is a transformed-based method where the noisy image undergoes multiple transformations during the denoising

process as shown in Figure 2.32. The noisy image is first converted to the wavelet domain from spatial domain using the forward DWT. A denoising algorithm is then performed on the image in the wavelet domain. Finally the image is converted back to the spatial domain from the wavelet domain using the inverse DWT. The resulting image should now contain significantly less noise.

Figure 2.32: Image Denoising Block Diagram



In 1994, Donoho and Johnstone developed a denoising technique based on the observation that since noise is uncorrelated and usually small with respect to the signal of interest, the wavelet coefficients that result from noise will be uncorrelated and most likely be small as well. Therefore in order to denoise an image, the image is first transformed to the wavelet domain, thresholding is performed on the wavelet coefficients to suppress wavelet coefficients corresponding to noise in the wavelet domain, and finally the image is converted back to the spatial domain to ultimately remove the noise.

2.7.1.1.1. Hard Thresholding

The simplest form of wavelet denoising is called "hard thresholding". In hard thresholding all the wavelet coefficients of the transformed image with absolute value less than a certain threshold are set to 0, or removed, while the other wavelet coefficients remain unchanged. A graph illustrating this is shown in Figure 2.33.

2.7.1.1.2. Soft Thresholding

A slightly more complicated form of wavelet denoising is called "soft thresholding", or "wavelet shrinkage". Soft thresholding is similar to hard thresholding in that wavelet coefficients with absolute value less than the threshold are set to 0, but differs by reducing the absolute value of, or shrinking, the remaining wavelet coefficients by the threshold value. In other words all the wavelet coefficients are affected by soft thresholding. A graph illustrating this is shown in Figure 2.34.

Figure 2.33: Hard Thresholding

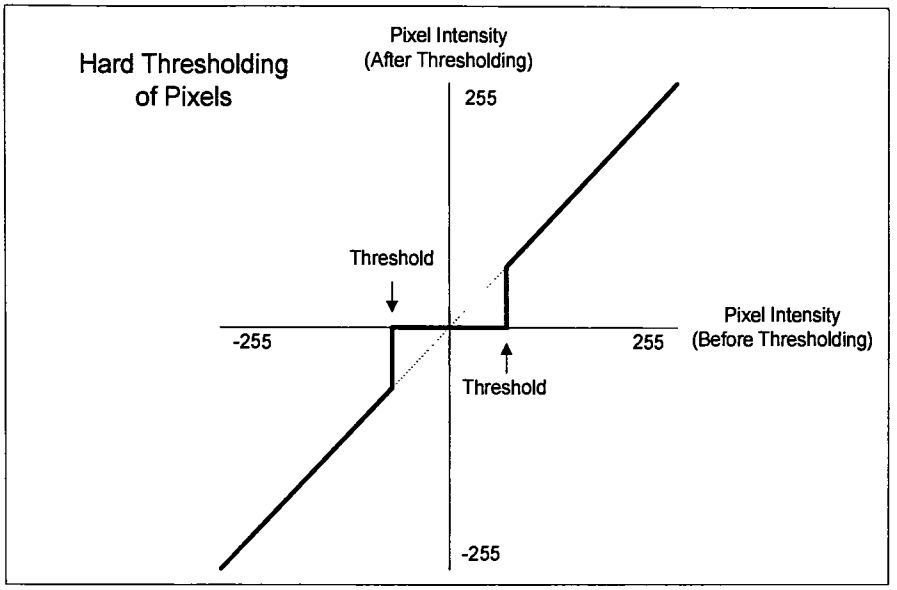
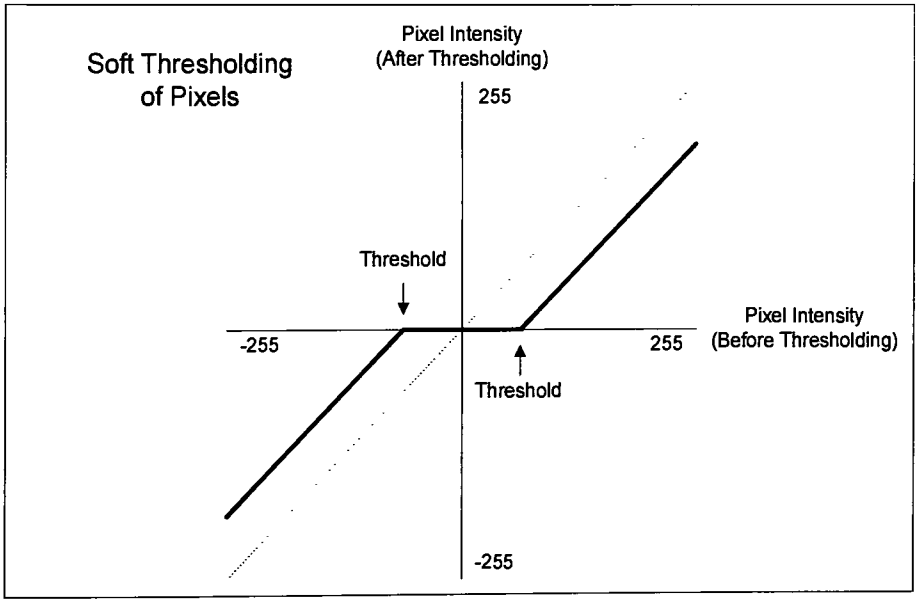


Figure 2.34: Soft Thresholding



The methods described above are not perfect because some parts of the signal of interest will also result in small coefficients which are indistinguishable from the noisy coefficients that are thrown away. Hard thresholding tends to preserve edges of the original image better than soft thresholding, but soft thresholding tends to remove noise better. Donoho determined the optimal

threshold value to be equal to $\sigma\sqrt{2 \ln N_s}$, where σ is the noise amplitude and N_s is the number of samples. [13, 14]

More complicated hybrid adaptive thresholding techniques can be used to improve the performance of wavelet denoising to better preserve edges and remove noise relative to basic hard or soft thresholding. The threshold value can vary based on scale and/or wavelet coefficient intensity. Both hard and soft thresholding can be used in combination across scales. The most complex technique that produces the best denoising results involves analyzing wavelet coefficient dependencies among subbands of the same and differing scales to determine which coefficients are noise-related. [15] It should be noted however that these hybrid techniques require significantly more overhead in terms of hardware complexity for extra control logic and pixel buffers as well as slower speed compared to the soft and hard thresholding methods described above.

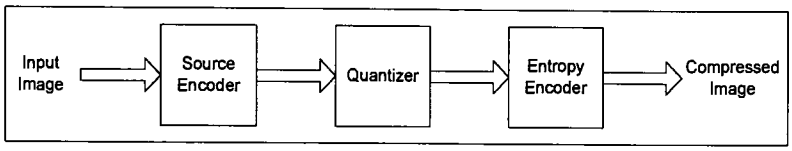
2.7.2. Image Compression

Digital images and media require enormous storage capacity and data transmission bandwidth that serve a large bottleneck in many applications. It is desirable and often necessary to represent a digital image in a compressed format to save space and bandwidth. Data compression attempts to remove redundancy and irrelevant information from the data source to produce a more compact representation. Digital image compression exploits the correlation of neighboring pixel values (spatial redundancy) and spectral bands (spectral redundancy) within an image to encode it using less information than the original. Digital video also makes use of correlation between adjacent frames of animation (temporal redundancy). Digital image compression can either be "lossless" or "lossy". In lossless compression the source image is compressed and decompressed in a manner that allows perfect reconstruction. In lossy compression the source image is compressed and decompressed in a manner that does not allow perfect reconstruction. The advantage of lossy compression over lossless is that a much higher compression ratio is possible at the expense of picture degradation, which may or may not be acceptable based on the particular application. [16]

2.7.2.1. Compression Scheme

A standard digital image compression encoder is composed of three steps: Source encoding, Quantization, and Entropy encoding. This process is shown in Figure 2.35. The decoder reverses the process. [16]

Figure 2.35: Image Compression Block Diagram



2.7.2.1.1. Source encoding

In the source encoding step a linear transform is used to decorrelate the image data into transformed coefficients. Commonly used source encoders are the DFT, DCT (Discrete Cosine Transform), and recently the DWT.

2.7.2.1.2. Quantization

The quantization step occurs in lossy compression and is the main source of compression in an encoder. The transformed coefficients are represented by a smaller number of bits by reducing their precision.

2.7.2.1.2. Entropy Encoding

During entropy encoding the quantized values are further compressed losslessly using their statistical characteristics into a code smaller than the stand-alone quantized values. Common entropy encoders are the Huffman encoder, arithmetic encoder, and run-length encoder.

2.7.2.2. JPEG2000

In 1988 the newly formed Joint Photographic Experts Group (JPEG) committee called for standardized still picture compression format. The new "JPEG" format was standardized in 1992 and was based off a close relative of the FT called the Discrete Cosine Transform (DCT) which was developed in 1974. In 1997 a new version of the JPEG still image compression standard called JPEG2000 was proposed to further enhance the capabilities of the aging JPEG standard. In 2000, part 1 of the JPEG2000 standard was produced with the major change being that the new standard would be use the DWT instead of the DCT. [17]

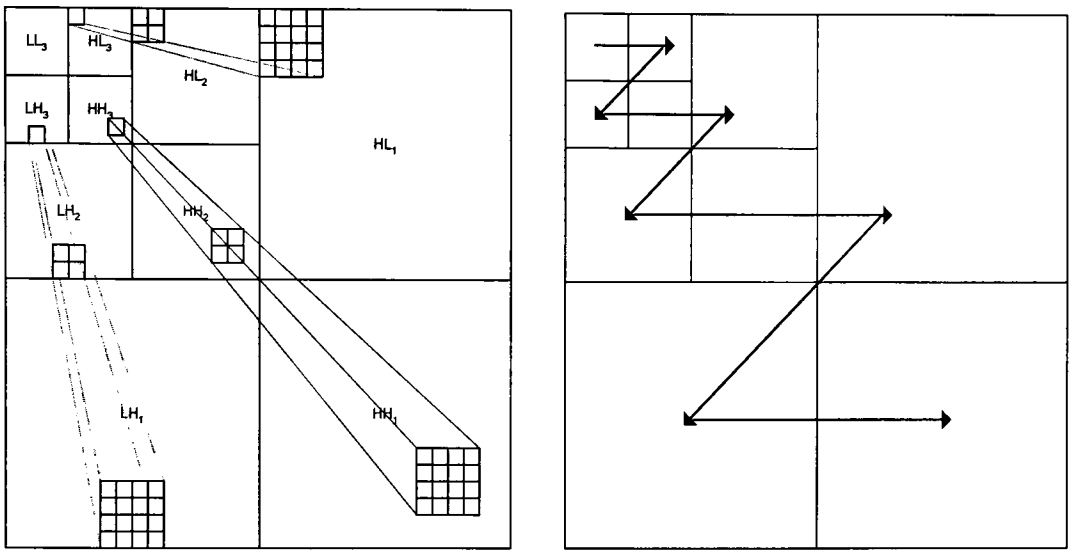
The DWT has recently been found to have many properties that make it more suitable for image compression that the DCT. DCT-based compression required the input image to be "blocked" and produced "blocking artifacts" at low bit rates. There is no need to block the input image with the DWT and therefore blocking artifacts are avoided. DWT-based compression is more robust under transmission and decoding errors and allows for progressive transmission of images. DWT-based compression is also more suitable for applications where scalability and tolerable degradation are important due to its multiresolution feature. [16]

Part 1 of the JPEG2000 standard provides one reversible lossless and one irreversible lossy compression path each using separate biorthogonal wavelets for the DWT source encoders. Part 2 of the JPEG2000 standard will allow for many more wavelets in either the reversible and irreversible path. The JPEG2000 standard supports convolution and lifting scheme methods for performing the DWT. The wavelet used for the reversible lossless DWT source encoder is the Spline 5/3 biorthogonal wavelet. The Spline 5/3 wavelet is a scaled version of the CDF 5/3 described earlier. In order to achieve lossless and reversible performance an integer-wavelet-transform version of the Spline 5/3 DWT is used by rounded coefficients to integer values. [7 (chapter 6)] The wavelet used for the irreversible lossy DWT source encoder is the CDF 9/7 biorthogonal wavelet mentioned earlier. Due to the floating point nature of the CDF 9/7 it is only used as an irreversible transform but due to its higher energy compaction than the Spline 5/3 it provides significantly better compression ratios. [7 (chapter 6)]

Each wavelet coefficient has four wavelet coefficients corresponding to its spatial location in a lower subband forming a quadtree structure. In 1993 Shapiro called this tree the "Zero Tree" structure of wavelet coefficients and developed his wavelet-based entropy encoding method called "Embedded Zerotree Wavelet" (EZW) algorithm shown in Figure 2.36. The EZW is based on the hypothesis that if a wavelet coefficient at a coarser scale is considered insignificant with respect to a given threshold then the wavelet coefficients in the finer scales corresponding to the same spatial location are also insignificant with respect to the same threshold. The EZW encodes the tree structure obtained starting a the most significant coarsest wavelet coefficients going down through the scales until the threshold is met. This allows encoding at a certain bit

rate to stop when the desired compression ratio is met. This also allows progressive transmission of a compression image by first decoding the most significant coarse wavelet coefficients down to the finer coefficients until the image is fully decoded, or decoding can stop when the decoded image is determined to be sufficient. [16]

Figure 2.36: Embedded Zero Tree



2.7.3. Other Applications

Some other useful digital image applications of the DWT are edge-detection, feature extraction, and classification. As mentioned earlier wavelets act as local edge detectors by representing edges in the image by large wavelet coefficients at the corresponding spatial location. Using this information and multiresolution analysis sharp or soft edges in an image can be determined easily based on wavelet coefficient values and spatial location. Feature extraction and image classification make use of the multiresolution property of wavelets to extract certain features from necessary scales of an image and classifying them.

2.8. Current Implementations of DWT

Recent commercial interest in the applications of the DWT have spawned both software and hardware incarnations of the DWT.

The advantages of performing the DWT in software versus hardware are the same as any other algorithm. The advantages of performing the DWT in software versus hardware are flexibility

and ease of integration. Many different wavelet families can be used with the same DWT algorithm, whether convolution-based or lifting scheme-based, by simply adjusting a few parameters. The software DWT can also be integrated into image and video software codecs easily. The amount of memory available to a software application is much more than that available to a hardware ASIC and therefore is not much of a concern in the development of a software algorithm. The advantages of performing the DWT in hardware versus software are speed and portability. A hardware-based DWT is much faster than the software-based DWT allowing for better real-time signal processing performance. A hardware-based DWT can also be implemented in a single-chip package for use in portable devices such as digital camera and scanners, whereas a software-based DWT requires a microprocessor to run the software. There are also advantages and disadvantages to an ASIC/FPGA hardware DWT implementation versus a DWT implemented on a DSP chip. While both are portable and the DSP implementation may be easier to develop, the ASIC/FPGA implementation could be more portable, run faster, use less power, and be cheaper if mass produced.

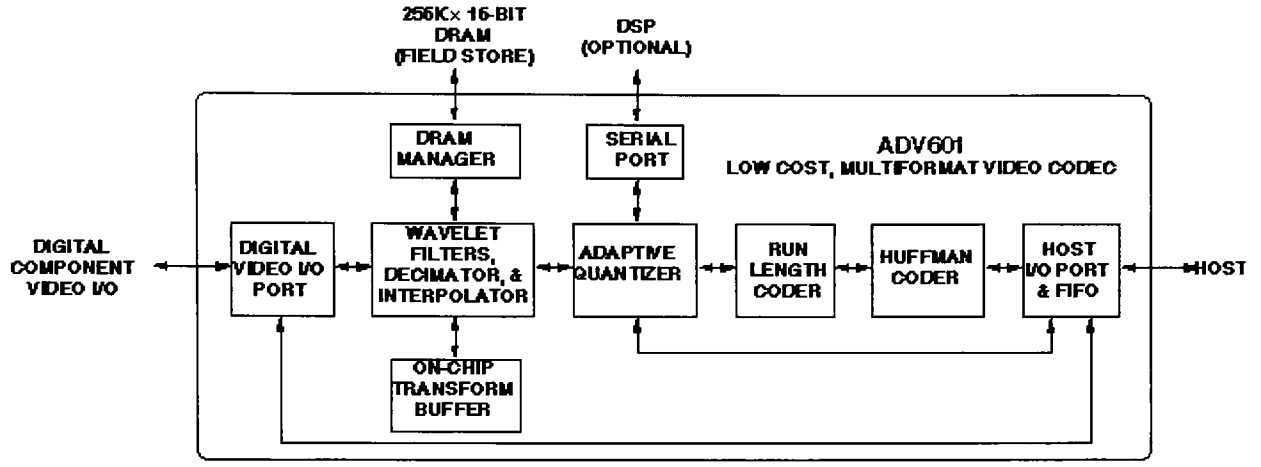
2.8.1. Software

Many software packages are available that implement the DWT. A very comprehensive software wavelet package is provided by MathWorks, Inc. called the Wavelet Toolbox. This toolkit is an open and customizable environment that allows the user to easily develop wavelet-based algorithms that builds upon the signal processing capabilities of MATLAB. This software package has a wide range of features that allows the user to perform the forward and inverse CWT and DWT on 1-D or 2-D signals such as audio and images using many different families of wavelets. This software package also provides common wavelet-based signal processing functions such as signal compression and denoising. [18]

2.8.2. Hardware

Analog Devices, Inc. developed the ADV601 chip which is an all-digital, CMOS VLSI, low-cost (~$35 each in lot of 10,000), real-time video compression and decompression solution using the DWT. This chip is capable of performing lossy and near-lossless compression up to 350:1 on video images using the CDF 9/7 convolution-based 2-D DWT as the transform coder implemented via FIR filters. The block diagram of this chip is shown in Figure 2.37. [19]

Figure 2.37: Analog Devices ADV601 Chip Block Diagram



CAST, Inc. in conjunction with Alma Technologies provides various DWT IP cores for commercial use. The JPEG2K_E is a JPEG2000 encoder core that is in compliance with the ISO/IEC 15444-1 JPEG2000 standard. The RC_2DDWT is a forward/inverse DWT core the implements the forward and inverse CDF 5/3 and CDF 9/7 2-D DWT using the lifting scheme that can be used in JPEG2000 and MPEG4 standards. The LB_2DFDWT is a line-based forward DWT core that uses the minimum required storage to perform the forward DWT. The core buffer uses the line-based DWT algorithm that buffers only the necessary lines of the image required to perform the DWT in the event the entire image cannot be buffered in memory or the image is unbounded as in satellite photos. The user customizes the core via VHDL generic parameters to fit their image dimensions, number of levels of transformation and memory requirements. Their final DWT related offering is the 97FG VHDL 9/7 DWT filter generator core that is a VHDL code generator that produces optimized and synthesizable VHDL descriptions of the 9/7 DWT filter which could then be used in the JPEG2000 or MPEG4 systems. [20]

Numerous papers have been published describing innovative architectures and hardware implementations of for performing the DWT. [21] and [22] describe efficient convolution-based architectures for the DWT using filter banks. [21] simulates a VLSI model of its architecture to obtain estimated hardware performance and suggests the hardware utilization of its architecture.

[22] implements its architecture on an Altera FPGA and suggests performance if it were to be implemented on an ASIC. [23] proposes a scalable architecture for performing many different DWTs using the lifting scheme. A behavioral VHDL model capable of performing the CDF 5/3 and CDF 9/7 forward and inverse DWT is implemented and estimates of hardware characteristics are derived. [24] compares both a convolution-based and a high-throughput pipelined lifting scheme-based DWT architecture. The lifting scheme architecture for the CDF 9/7 DWT is translated to RTL Verilog and synthesized to CMOS technology to obtain hardware characteristics. [25] proposes a VLSI architecture for performing the CDF 9/7 Integer Wavelet Transform. A structural VHDL model of the CDF 9/7 Integer Wavelet Transform is synthesized and hardware characteristics are obtained. [26] expands upon [25] to propose a power efficient architecture for performing the DWT using the lifting scheme and techniques for improving performance at the cost of increased power consumption and vice-versa. An RTL VHDL architecture of the CDF 9/7 DWT is synthesized to Xilinx implemented the CDF 9/7 DWT in RTL VHDL and synthesized to a Xilinx FPGA technology to obtain hardware characteristics. [27] describes architectures used to improve 2-D DWT performance using the lifting scheme. This paper proposes certain techniques that allow memory savings or performance increases by performing row and column processing simultaneously and/or multiple levels of DWT decomposition simultaneously in the 2-D DWT scenario. [28] proposes a "folded" architecture of the pipelined lifting-based DWT implementation, reusing multipliers to reduce hardware area and improve hardware utilization. The use of shift-add operations in place of regular multipliers to reduce hardware area is also proposed. [29] provides analysis on hardware characteristics of convolution-based and lifting scheme-based DWT each using both regular and shift-add multipliers for the CDF 9/7. This paper also analyzes the effect of using finite precision coefficients. [30] and [31] provide unique scalable architectures using the line-based lifting scheme method similar to that employed in this thesis to reduce the memory requirements for performing the 2-D DWT. It should be noted that none of the papers described lifting scheme-based architectures have implemented or tested their architectures on actual hardware.

Chapter 3. Design and Implementation of DWT Core

This chapter describes the unique features of the DWT core developed in this thesis. The design of the DWT core and its implementation is discussed in detail.

3.1. DWT Core Features

The DWT core developed in this thesis has many attractive features. These features include hardware flexibility, versatility, portability and efficiency.

3.1.1. Hardware flexibility

The DWT core performs the DWT using the lifting scheme. The DWT core is preconfigured with a set of lifting and scaling steps to define the user-defined DWT before run-time to generate the appropriate hardware automatically. Therefore in order to configure the DWT core for a particular wavelet, lifting step extraction must be performed to obtain the necessary lifting and scaling steps prior to configuration. Any wavelet that can be factored into lifting steps can be implemented by this DWT core. Any number of lifting steps can be defined, but only two scaling steps are allowed at maximum. The lifting/scaling steps are defined by real-number signed coefficients. In addition each lifting step can use a rounding/ceiling/floor/truncation function as required by the particular wavelet. It is even possible to use modified or non-standard lifting and scaling steps to create a new wavelet. The inputs and outputs of the DWT core are also preconfigured to define such things as maximum image width/height, maximum levels of decomposition, and size of memory data/address bus. The pixel width can be configured which determines the size of the input/output samples/coefficients as well as the datapath width for the lifting-based arithmetic operations.

3.1.2. Versatility

After the DWT core is configured and implemented in hardware, many options are available at run-time. The DWT core can perform both the forward and inverse DWT on an image of any height or width. The DWT core can also perform the DWT on smaller portions of the image. This is particularly useful for when the DWT needs to be applied to one or more tiles of an image as is the case with JPEG2000. The DWT can perform the DWT for any level of decomposition

allowable by the image size. For example an image of 512 x 512 pixels can be decomposed up to 9 times.

3.1.3. Portability

The DWT core is also portable. Since the DWT core is written in a standard hardware description language using low-level constructs it can be ported to most types of hardware such as FPGAs and ASICs.

3.1.4. Efficiency

The DWT core has many attributes that make it highly efficient.

The DWT core performs the DWT on an image completely in-place requiring no auxiliary memory. The memory used to store the image is overwritten during the transformation process thus eliminating the need for extra memory such as line buffers or image buffers.

The DWT core uses less memory bandwidth than other DWT cores, either convolution-based or lifting scheme-based. DWT cores using convolution require two passes per row/column of pixels for each 1-D DWT that is performed. DWT cores using the lifting scheme require two or more passes. This wavelet core implements the "line-based" DWT similar to that described in [32] that requires only one pass per row/column. As a result, however, each lifting step requires separate multiplication units.

Since the lifting step coefficients do not change at run time, the DWT core uses fixed-point shift-add multiplier hardware that is specifically tuned for each lifting step coefficient instead of using regular floating-point multipliers for real number coefficients. These units automatically generate the necessary hardware to perform multiplication of an integer number by any precision signed floating point number. These units are faster and use less area/power than general purpose floating-point multipliers.

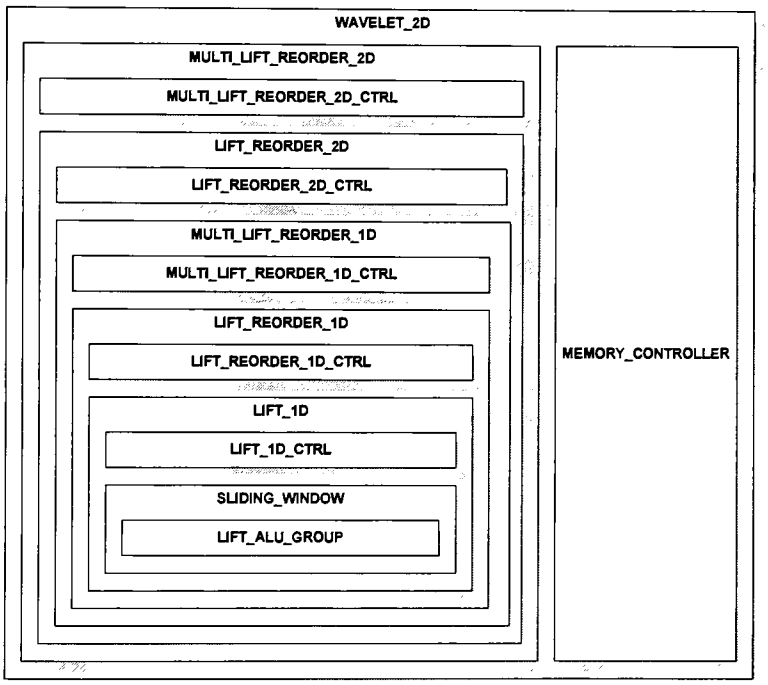
The inverse DWT using the lifting scheme is symmetrical and therefore performed by simply reversing the lifting steps in the forward DWT. The DWT core takes advantage of this by simply

reusing the ALU components from the forward DWT by rerouting the data path in the reverse direction and changing the additions to subtractions and vice-versa. As a result, the extra hardware overhead in terms of control and routing for the reverse DWT is minimal.

3.2. DWT Core Design

The component diagram of the DWT core is shown in the Figure 3.1.

Figure 3.1: DWT Core Component Diagram

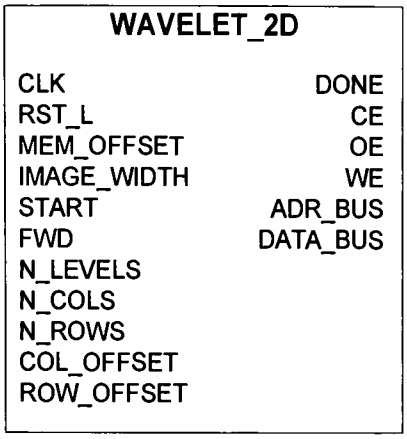


The functionality, internal components and interface of each hardware module within the DWT core are described below top-down.

3.2.1. WAVELET_2D

The WAVELET_2D module is the actual DWT core. It interconnects the MULTI_LIFT_REORDER_2D unit to the MEMORY_CONTROLLER unit. The interface for the DWT core is shown in Figure 3.2.

Figure 3.2: DWT Core Interface



3.2.2. MEMORY_CONTROLLER

The MEMORY_CONTROLLER module is the memory controller. It provides the synchronous read/write interface to asynchronous SRAM memory. This module translates the 2-D column and row address format to the 1-D memory address bus format using adders and multipliers. This module coordinates read and write access that share a single data bus. The state machine is shown in Figure 3.3.

Figure 3.3: MEMORY_CONTROLLER State Diagram

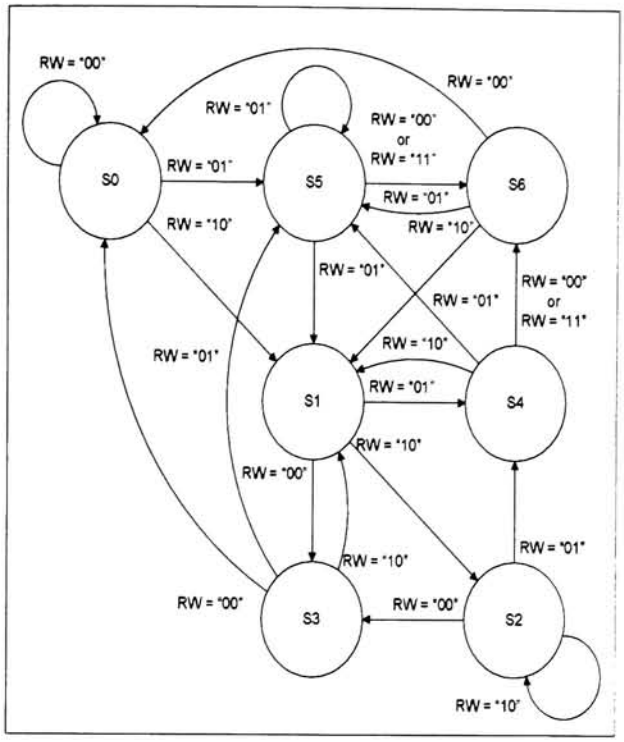


Table 3.1: MEMORY_CONTROLLER State Output Table

Output	Value	State
CE_out	'0'	S1, S2, S4, S5
	'1'	All other states
OE_out	'0'	S1, S2
	'1'	All other states
WE_out	'0'	S4, S5
	'1'	All other states
RD_PIXEL	DATA_BUS_inout	S1
	TEMP_RD_PIXEL	All other states
DATA_BUS_inout	TEMP_WR_PIXEL	S4, S5, S6
	High impedance "ZZ...ZZ"	All other states

3.2.3. MULTI_LIFT_REORDER_2D

The MULTI_LIFT_REORDER_2D module performs the 2-D forward/inverse DWT for N levels of decomposition of an image. It uses control logic and the LIFT_REORDER_2D unit to accomplish this. The MULTI_LIFT_REORDER_2D_CTRL module is the control logic for the MULTI_LIFT_REORDER_2D unit. It is a state machine controller used to control the LIFT_REORDER_2D unit to perform an N-level 2-D DWT decomposition using multiple 1-level decompositions recursively applied to the LL subbands of each level. The state-machine is shown in Figure 3.4..

Figure 3.4: MULTI_LIFT_REORDER_2D_CTRL State Diagram

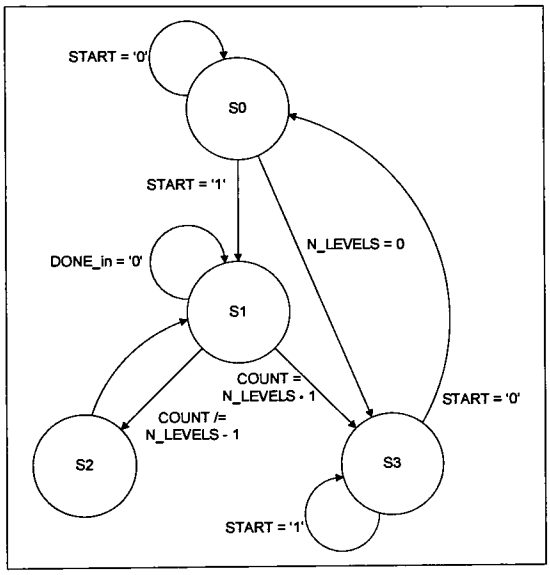


Table 3.2: MULTI_LIFT_REORDER_2D_CTRL State Output Table

Output	Value	State
DONE	'1'	S3
	'0'	All other states
RST_L_out	'0'	S2
	'1'	All other states
START_out	'1'	S1
	'0'	All other states
INC_CTL	'1'	S2
	'0'	All other states
SR_CTL	"11"	S0
	"01"	S2 (FWD='1')
	"10"	S2 (FWD='0')
	"00"	All other states

3.2.4. LIFT_REORDER_2D

The LIFT_REORDER_2D module performs the 2-D forward/inverse DWT for one level of decomposition of an image. It uses control logic and the MULTI_LIFT_REORDER_1D unit to accomplish this. The LIFT_REORDER_2D_CTRL module is the control logic for the LIFT_REORDER_2D unit. It is a state machine controller used to control the MULTI_LIFT_REORDER_1D unit to perform a 1-level 2-D DWT decomposition. To do this the MULTI_LIFT_REORDER_1D unit is first applied horizontally to the rows of an image and then vertically to the columns of the image for the forward DWT. For the inverse DWT it is the opposite. The state-machine is shown in Figure 3.5.

Figure 3.5: LIFT_REORDER_2D_CTRL State Diagram

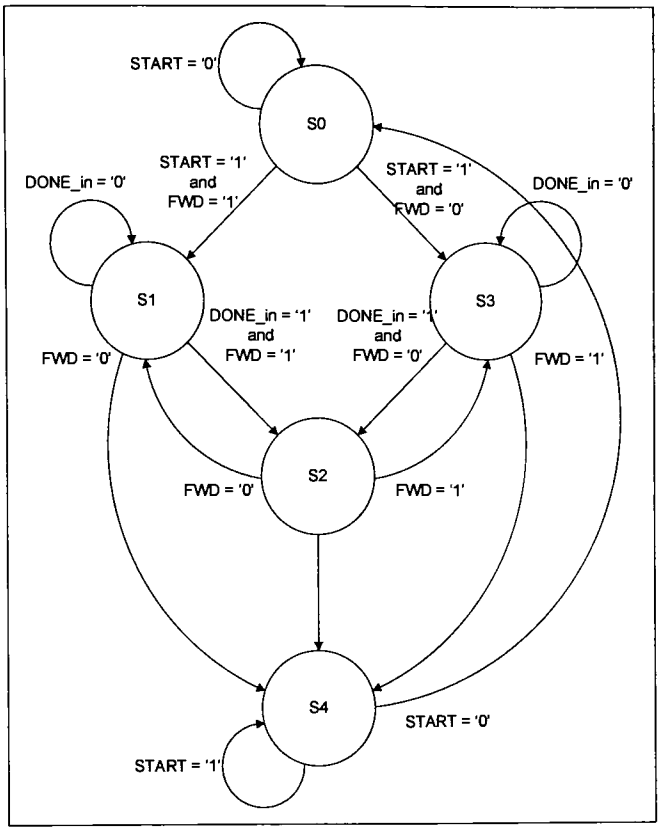


Table 3.3: LIFT_REORDER_2D_CTRL State Output Table

Output	Value	State
DONE	'1'	S4
	'0'	All other states
RST_L_out	'0'	S0, S2
	'1'	All other states
START_out	'1'	S1, S3
	'0'	All other states
N_PIXELS_out	N_COLS	S1
	N_ROWS	All other states
N_LINES_out	N_ROWS	S1
	N_COLS	All other states
RD_COL_ADR_out	RD_ADR_in	S1
	RD_WR_LINE_ADR_in	All other states
WR_COL_ADR_out	WR_ADR_in	S1
	RD_WR_LINE_ADR_in	All other states
RD_ROW_ADR_out	RD_WR_LINE_ADR_in	S1
	RD_ADR_in	All other states
WR_ROW_ADR_out	RD_WR_LINE_ADR_in	S1
	WR_ADR_in	All other states

3.2.5. MULTI_LIFT_REORDER_1D

The MULTI_LIFT_REORDER_1D module performs the 1-D forward/inverse DWT on N lines (rows/columns) of an image. It uses control logic and the LIFT_REORDER_1D unit to accomplish this. The MULTI_LIFT_REORDER_1D_CTRL module is the control logic for the MULTI_LIFT_REORDER_1D unit. It is a state machine controller used to control the LIFT_REORDER_1D unit to perform the 1-D DWT on N lines. The state-machine is shown in Figure 3.6.

Figure 3.6: MULTI_LIFT_REORDER_1D_CTRL State Diagram

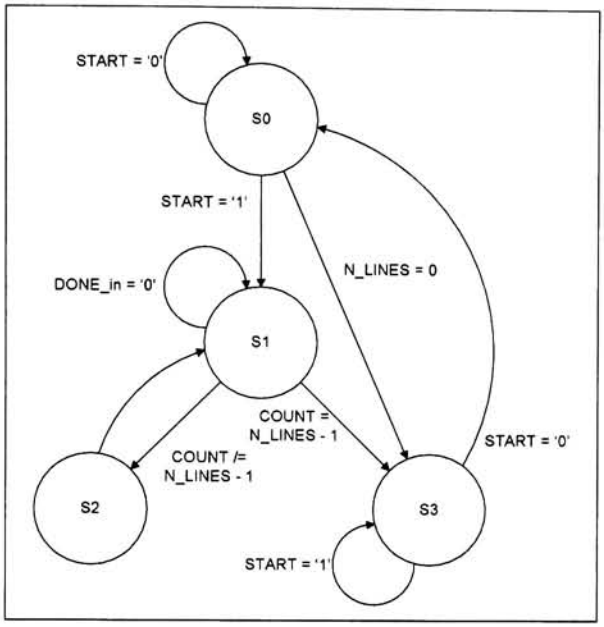


Table 3.4: MULTI_LIFT_REORDER_1D_CTRL State Output Table

Output	Value	State
DONE	'1'	S3
	'0'	All other states
RST_L_out	'0'	S2
	'1'	All other states
START_out	'1'	S1
	'0'	All other states
INC_CTL	'1'	S2
	'0'	All other states

3.2.6. LIFT_REORDER_1D

The LIFT_REORDER_1D module performs the 1-D forward/inverse DWT with reordering on one line of an image. The LIFT_REORDER_1D_CTRL module is the control logic for the LIFT_REORDER_1D unit. It is a state machine controller used to control the LIFT_1D unit to perform the lifting scheme and reordering on one line. For the forward DWT the lifting scheme is

performed followed by reordering. For the inverse DWT it is the opposite. The state-machine is shown in Figure 3.7.

Figure 3.7: LIFT_REORDER_1D_CTRL State Diagram

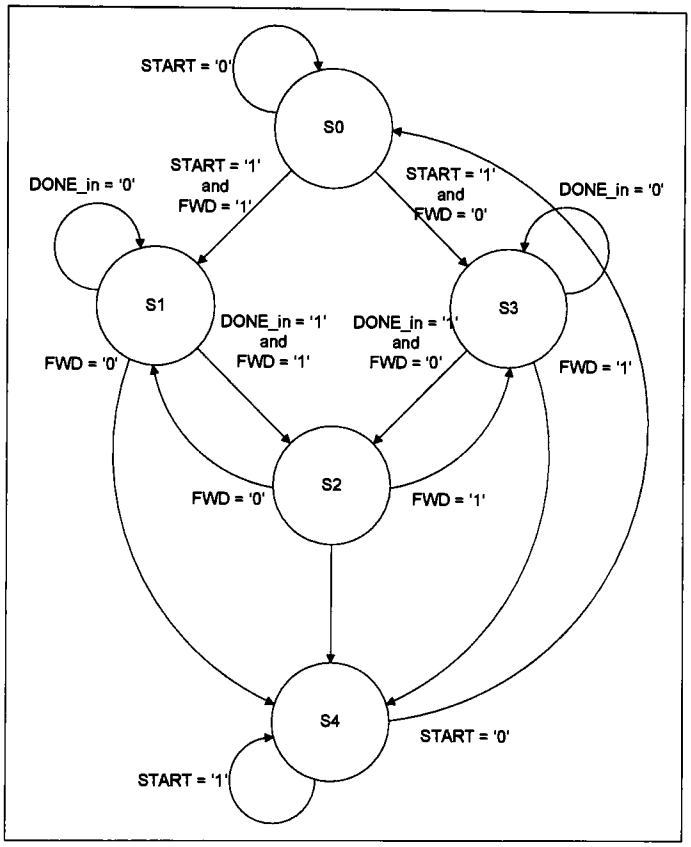


Table 3.5: LIFT_REORDER_1D_CTRL State Output Table

Output	Value	State
DONE	'1'	S4
	'0'	All other states
RST_L_out	'0'	S0, S2
	'1'	All other states
START_out	'1'	S1, S3
	'0'	All other states
LIFT_out	'1'	S1
	'0'	All other states

3.2.7. LIFT_1D

The LIFT_1D module performs either the lifting scheme or reordering on one line of an image.

3.2.7.1. Sliding Window Method

The standard lifting scheme method mentioned earlier requires a row of pixels to be processed multiple times, once for each lifting and scaling step. Therefore the CDF 5/3 DWT requires two passes of each row and the CDF 9/7 requires six passes of each row.

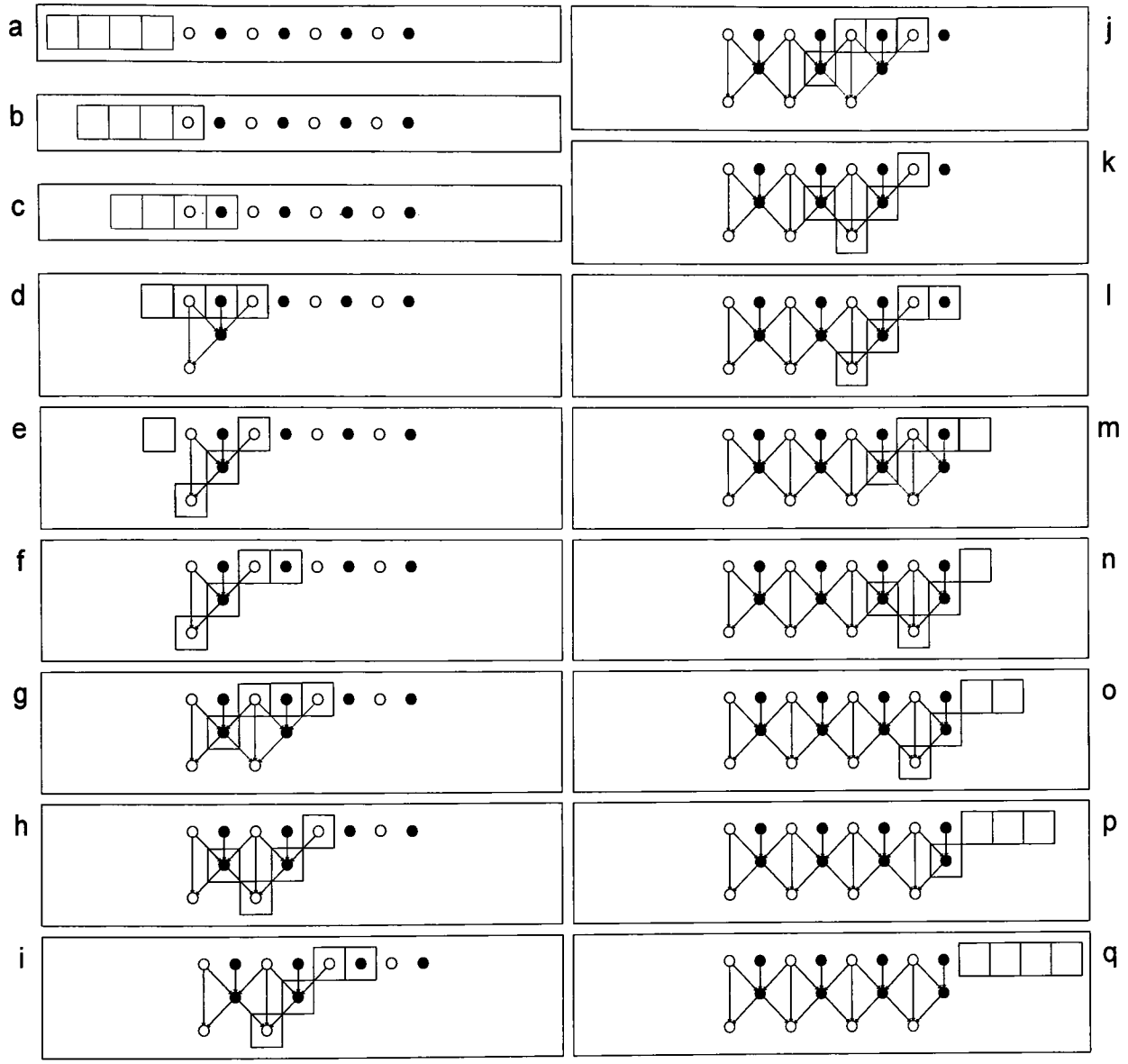
The algorithm presented below, to be referred to in this thesis as the "Sliding Window Method," is similar to the line-based techniques mentioned earlier. This algorithm was developed so that regardless of the number of lifting steps there is only a single pass of each row of pixels to reduce the memory I/O bandwidth. The standard lifting scheme calculates the scaling and wavelet coefficients by incrementally overwriting the original pixel value with the intermediate lifting step results until the final scaling and wavelet coefficient values are obtained. Multiple passes of each row is necessary to eliminate the need to intermediate storage buffers.

The concept behind this method is to buffer the necessary pixels required to calculate the final scaling and wavelet coefficients. From analysis of the lifting scheme pixel dependency graph it has been determined that the lifting scheme can be implemented with the use of minimal pixel buffers the lifting scheme requiring only a single pass of each row can be implemented. Rather than storing the results from the intermediate lifting steps the intermediate values are buffered and used to calculate the final scaling and wavelet coefficients which are then used to overwrite each of the original pixels. It has been determined that for a wavelet with N lifting steps an N+2 size pixel buffers are required regardless of whether or not scaling steps are present. Therefore, the CDF 5/3 with two lifting steps requires a four-pixel buffer and the CDF 9/7 with four lifting steps requires a six-pixel buffer.

3.2.7.1.1. CDF 5/3 Sliding Window Method

Figure 3.8 illustrates the CDF 5/3 forward DWT on a row of eight pixels using the sliding window method.

Figure 3.8: Forward CDF 5/3 DWT of Row of Pixels using Sliding Window Method



First the sliding window buffer reads in two pixels from memory into a four-pixel storage buffer as shown in Figure 3.8a, b, and c. After the third pixel is read into the buffer the first set of scaling and wavelet coefficients are calculated in Figure 3.8d. The newly calculated scaling and wavelet coefficients are stored into the buffer in Figure 3.8e. Another two pixels are read into the buffer filling the buffer in Figure 3.8f and g. As the pixels are read from memory and shifted into the head of the filled buffer, the scaling and wavelet coefficients are shifted out from the tail of the buffer are written back to memory. This process is repeated until the entire row has been

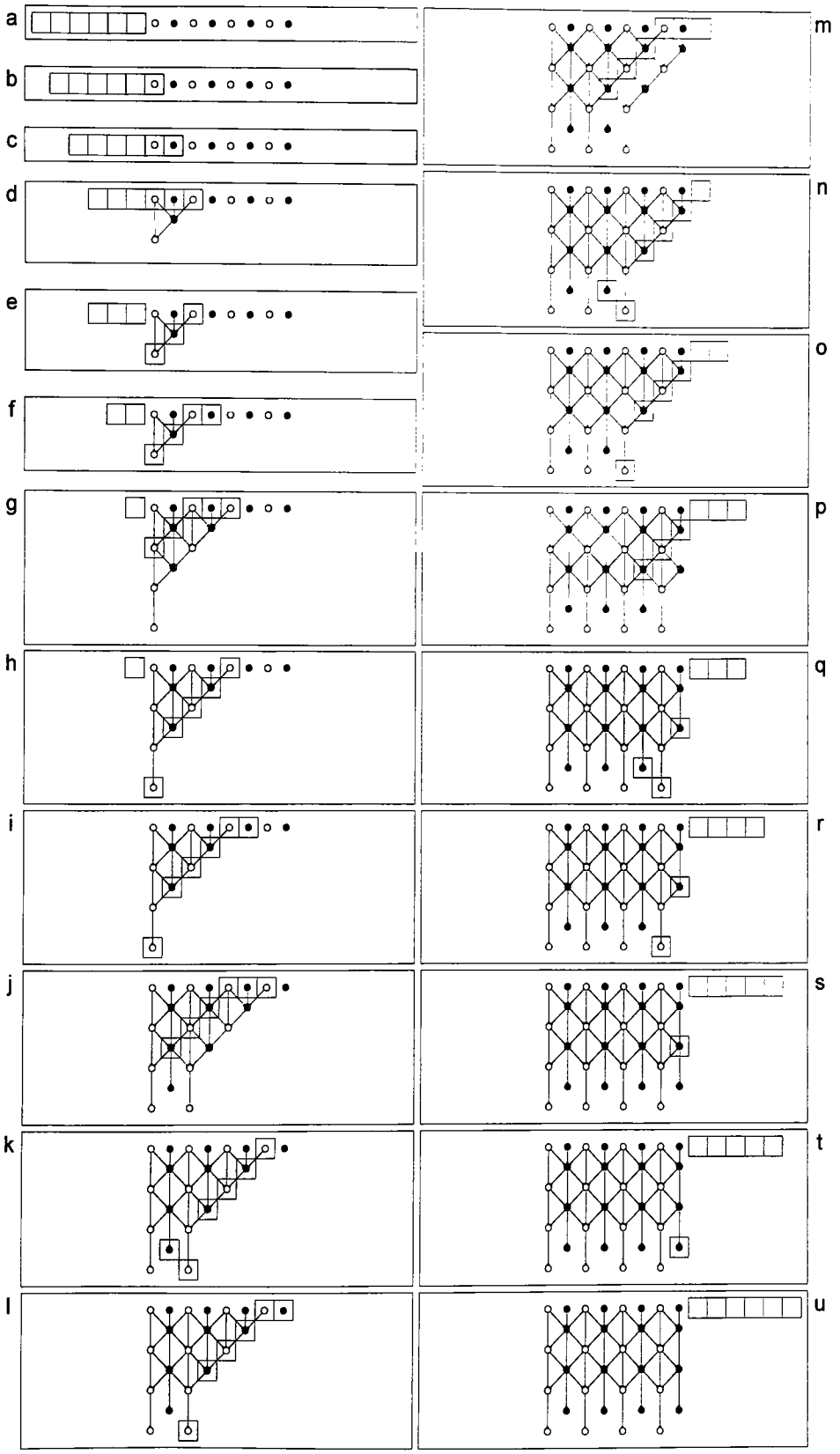
processed in Figure 3.8h through q. The buffer has written all of its pixels back into memory and is now ready to process another row.

3.2.7.1.2. CDF 9/7 Sliding Window Method

Figure 3.9 illustrates the CDF 9/7 forward DWT on a row of eight pixels using the sliding window method.

First the sliding window buffer reads in two pixels from memory into a six-pixel storage buffer as shown in Figure 3.9a, b, and c. After the third pixel is read into the buffer the intermediate results from the first two lifting steps are calculated in Figure 3.9d and e. Another two pixels are read into the buffer. After the fifth pixel is read into the buffer the first scaling coefficient is calculated by passing through a lifting alu and then a scaling alu as well as the second set of intermediate pixel results shown in Figure 3.9f and g. The newly calculated scaling coefficient and intermediate results are stored into the buffer in Figure 3.9h. Another two pixels are read into the buffer filling the buffer. As the pixels are read from memory and shifted into the head of the buffer, the scaling and wavelet coefficients are shifted out from the tail of the buffer are written back to memory in Figure 3.9i. Note, the wavelet coefficients undergo their scaling step before being written back to memory in Figure 3.9j. The process is repeated until the entire row has been processed shown in Figure 3.8k through u.

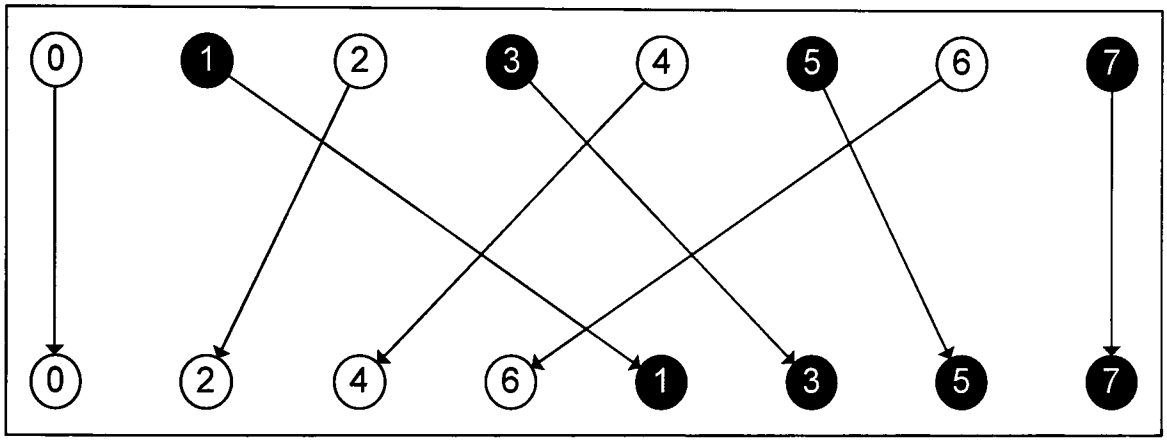
Figure 3.9: Forward CDF 9/7 DWT of Row of Pixels using Sliding Window Method



3.2.7.2. Reordering

Reordering a row of pixels to group the scaling and wavelet coefficients together after the DWT has been performed can be a very costly operation in terms of memory bandwidth and execution time. The goal is to replace a line of pixels having the scaling (even pixels) and wavelet (odd pixels) coefficients interlaced such that the two sets of coefficients are grouped together. As shown in the figure 3.10 the even pixels are copied to the left side of the row while the odd pixels are copied to the right side of the row. Figure 3.10 shows a row of 8 pixels labeled by their addresses being reordered.

Figure 3.10: Reordering of Scaling and Wavelet Coefficients within a Row



There are many possible methods to reorder the pixels. The most basic method is to simply copy the entire row of pixels into the correct positions in a separate line buffer and then copy them back to the original memory location. While this method is straight forward it requires additional memory to buffer the entire row of pixels. Furthermore, for a row of N pixels there are N memory reads from the original memory location, N memory writes to the reordered buffer, N memory reads from that buffer and then N memory writes back to the original memory location. This method requires a total of 4N memory accesses to reorder a row of N pixels plus the additional N-pixel storage buffer. Many other sort methods can be employed requiring no intermediate storage buffer but at the cost of significant memory read and write overhead and/or computational complexity. The algorithm described below was developed as an efficient solution to address this problem.

This reordering algorithm requires a single-pixel storage buffer, a status register, a decoder, priority encoder and an ALU for address calculation. Since the sliding window buffer is not used during reordering its memory elements can be used for reordering reducing the hardware requirements. Thus, the pixel storage buffer can be shared with the sliding window buffer as well as the status register.

The status register keeps track of which pixels still need to be reordered. Due to the symmetrical nature of the reordering only the status of the pixels of one half of the row need recorded. Therefore for an image of maximum column/row size of N pixels a status register of size N/4 bits is required, however, only a portion of the status bits are used depending on the number of pixel to be reordered. As each odd pixel with address less than N/2 (left side of the row) is reordered a decoder is used to access the appropriate status bit in the status register given the current pixel's address. The following equation is used to determine the status bit to be cleared from the address.

$$\text{Status Bit} = \left\lfloor \frac{\text{Address}}{2} \right\rfloor$$

Therefore, the status bit of the first odd pixel 1 is status bit 0, the status bit for the second odd pixel 3 is status bit 1, etc up to the N/2 odd pixel. That bit is then cleared indicating that the pixel has been reordered. When reordering is attempted on a pixel whose status bit has already been cleared indicating it has been reordered, a priority encoder is used to find the address of a lower-address non-reordered odd pixel from the status register.

$$\text{Address} = (2 \times \text{Status Bit}) + 1$$

Therefore, the status bit 0 indicates the reordered status of first odd pixel 1, status bit 1 indicates the reordered status of second odd pixel 3, etc up to the N/2 odd pixel. Once all the status register bits are clear reordering has completed. The addresses of the reordered pixels are calculated using the following equations:

Forward DWT, scaling (even) coefficients:

$$\text{New Address} = \frac{\text{Old Address}}{2}$$

Forward DWT, wavelet (odd) coefficients:

$$\text{New Address} = \left\lfloor \frac{\text{N} + \text{Old Address}}{2} \right\rfloor$$

Inverse DWT, scaling (left) coefficients:

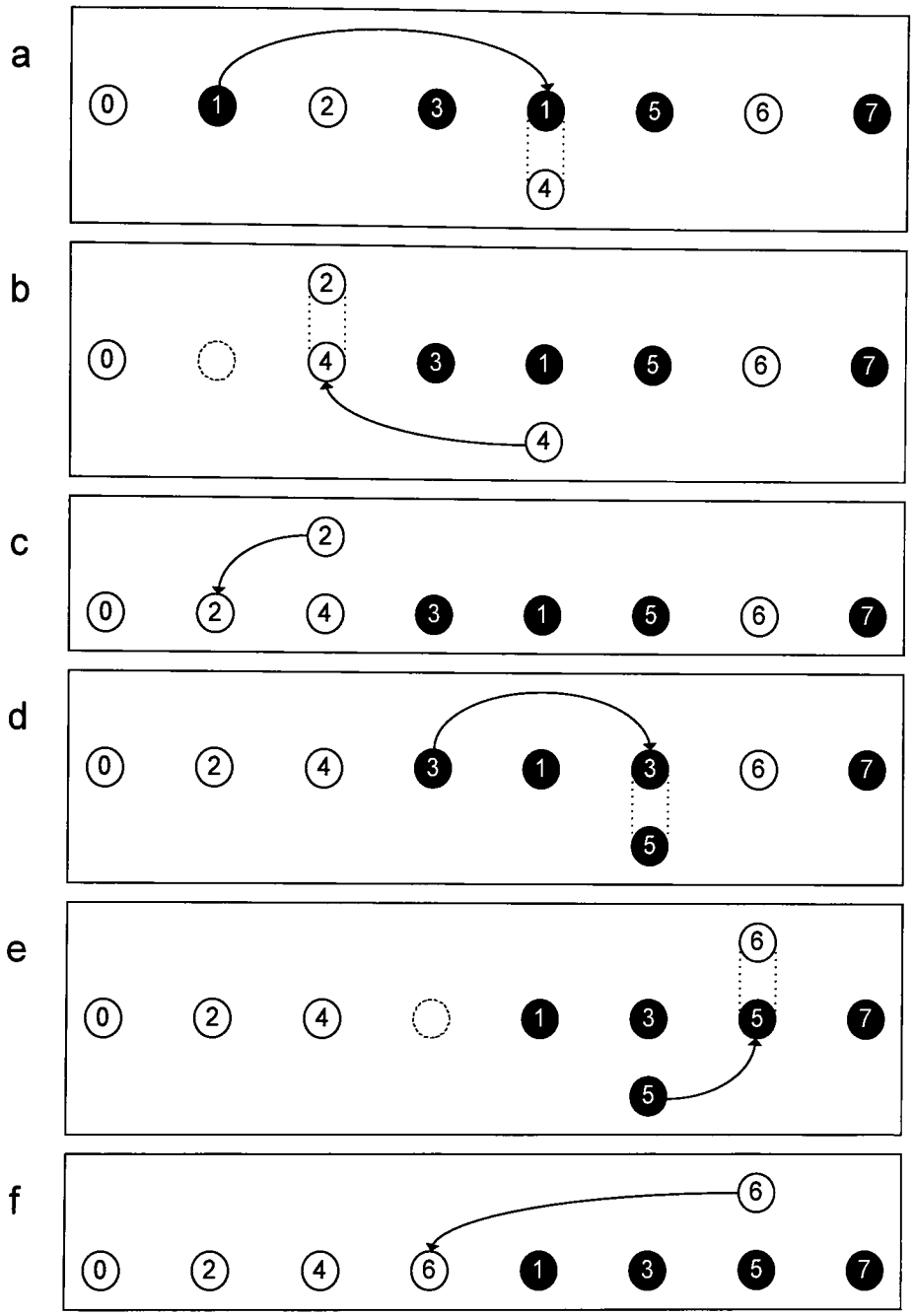
$$\text{New Address} = 2 \times \text{Old Address}$$

Inverse DWT, wavelet (right) coefficients:

$$\text{New Address} = (2 \times \text{Old Address}) - (\text{N} - 1)$$

The time and hardware required for address calculation is minimal since the calculations are mostly multiply/divide by 2 requiring only simple left/right shift operations in addition to at most one addition. Figure 3.11 illustrates the reordering algorithm applied to a row of 8 pixels.

Figure 3.11: Reordering Algorithm for Scaling and Wavelet Coefficients within a Row



The status register consists of N/4=8/4=2 bits. Bit 0 of the status register represents pixel 1 and the bit 0 represents pixel 3. Reordering begins with the first odd pixel in the row since the first even pixel does not get moved during the reordering. The status bit for pixel 1 (bit 0) is not yet cleared so pixel 1 moves to address 4 according to the equation below.

$$\text{New Address} = \left\lfloor \frac{\text{N} + \text{Old Address}}{2} \right\rfloor = \left\lfloor \frac{8+1}{2} \right\rfloor = 4$$

The status bit for pixel 1 (bit 0) is now cleared indicating it has been reordered. Pixel 4 is copied into the pixel buffer before pixel 1 overwrites it as shown in Figure 3.11a. Pixel 4 does not have a status bit and therefore pixel 4 moves to address 2 according to the equation below.

$$\text{New Address} = \frac{4}{2} = 2$$

No status bits are cleared since pixel 4 does not have a status bit. Pixel 2 is copied into the pixel buffer before pixel 4 overwrites it in Figure 3.11b. Pixel 2 does not have a status bit and therefore pixel 2 moves to address 1 in Figure 3.11c according to the equation below.

$$\text{New Address} = \frac{2}{2} = 1$$

No status bits are cleared since pixel 2 does not have a status bit. Since the status bit for pixel 1 (bit 0) has already been cleared indicating it has already been reordered the priority encoder is used to determine the next pixel to reorder. Status bit 1 indicates that pixel 3 has not yet been reordered and therefore that is the next pixel to be reordered. Pixel 3 moves to address 5 according to the equation below.

$$\text{New Address} = \left\lfloor \frac{\text{N} + \text{Old Address}}{2} \right\rfloor = \left\lfloor \frac{8+3}{2} \right\rfloor = 5$$

The status bit for pixel 3 (bit 1) is now cleared indicating it has been reordered. Pixel 5 is copied into the pixel buffer before pixel 3 overwrites it in Figure 3.11d. Pixel 5 does not have a status bit and therefore pixel 5 moves to address 6 according to the equation below in Figure 3.11e

$$\text{New Address} = \left\lfloor \frac{\text{N} + \text{Old Address}}{2} \right\rfloor = \left\lfloor \frac{8+5}{2} \right\rfloor = 6$$

No status bits are cleared since pixel 6 does not have a status bit. Pixel 6 does not have a status bit and therefore pixel 6 moves to address 3 in Figure 3.11f according to the equation below.

$$\text{New Address} = \frac{6}{2} = 3$$

No status bits are cleared since pixel 6 does not have a status bit. Since the status bit for pixel 3 (bit 1) has already been cleared indicating it has already been reordered the priority encoder is used to determine the next pixel to reorder. Since all the status bits have been cleared reordering has

completed. The reordering for the inverse DWT is performed in exactly the same manner with the exception of using different address calculation equations.

This algorithm is very time and resource efficient. For a row of N pixels only N memory reads and N memory writes are necessary for a total of 2N memory accesses, half of the alternative algorithms mentioned above. In addition the reordering is in-place requiring no additional pixel row buffers.

The REORDER_ADR_CALC unit is the arithmetic unit used to calculate the reorder address of the pixel. The REORDER_STATUS_UNIT unit is used to keep track of which pixels have or have not been reordered.

The LIFT_ 1D_CTRL module is the control logic for the LIFT_1D unit. It is a state machine controller used to control the SLIDING_WINDOW buffer unit either to perform the lifting scheme or reorder a set of pixels. Since only one operation or the other is used at one time, the two separate state machines share the same hardware. The state-machine for the lifting scheme is shown in Figure 3.12.

Figure 3.12: LIFT_1D_CTRL Lifting Scheme State Machine

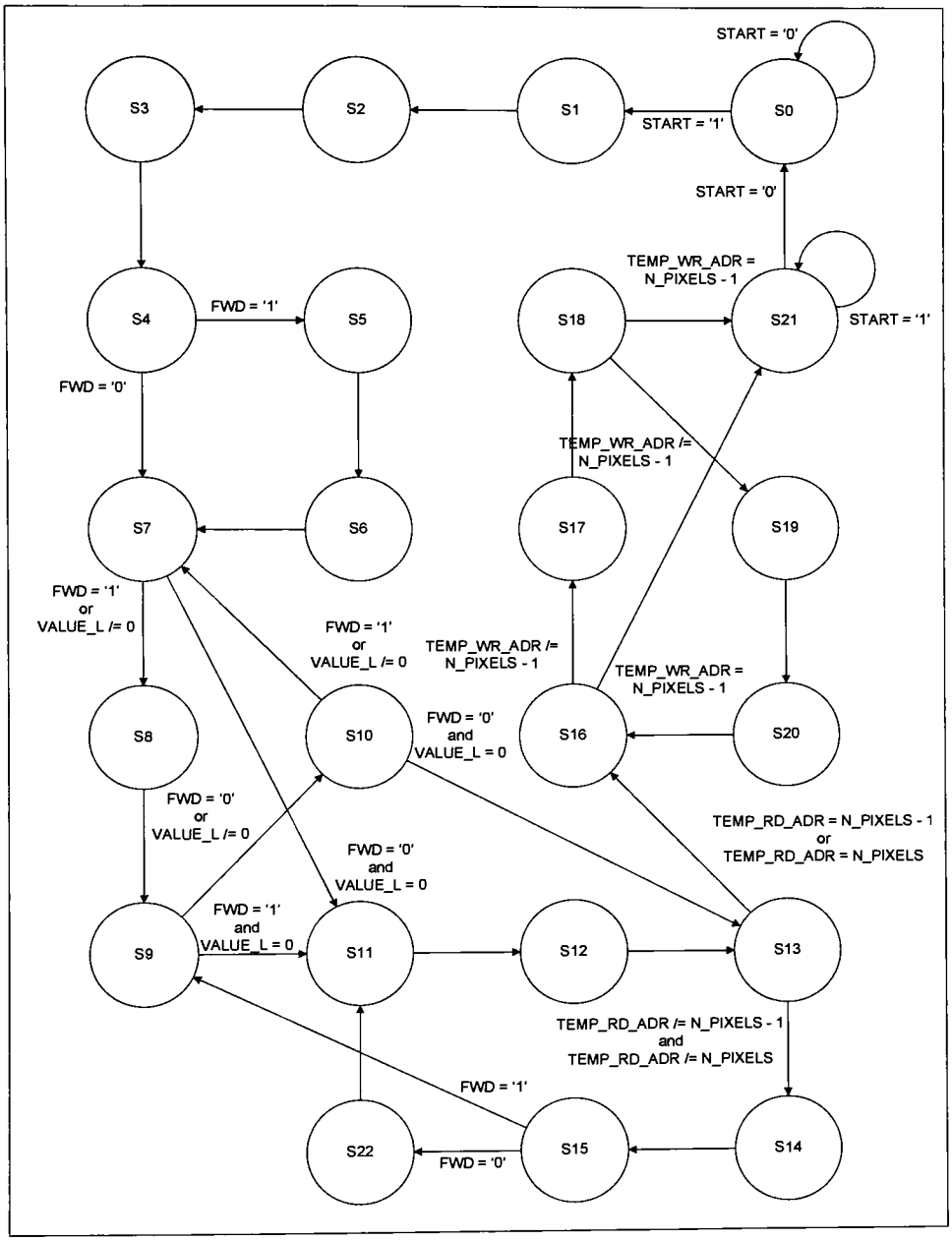


Table 3.6: LIFT_1D_CTRL Lifting Scheme State Output Table

Output	Value	State
DONE	'1'	S21
	'0'	All other states
CTL_L	'1'	S8
	'0'	All other states
CTL_R	'1'	S16
	'0'	All other states
INC_RD_CTL	'1'	S2, S4, S6, S8, S10, S11, S14
	'0'	All other states
INC_WR_CTL	'1'	S12, S15, S17, S19
	'0'	All other states
SHIFT_MODE_out	"11"	S7, S13, S20
	"10"	S2, S4, S6, S8, S10, S12, S15, S17, S19
	"00"	All other states
RW_out	"01"	S11, S14, S16, S18
	"10"	S1, S3, S5, S7, S9, S13, S22
	"00"	All other states

The state-machine for reordering is shown in Figure 3.13.

Figure 3.13: LIFT_1D_CTRL Reordering State Machine

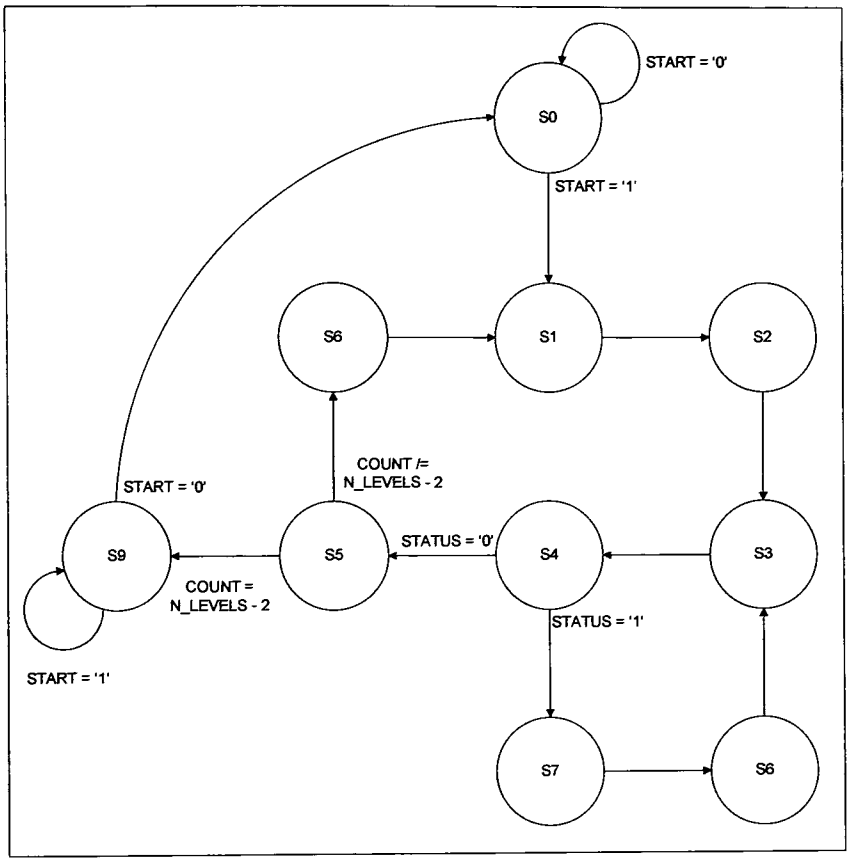


Table 3.7: LIFT_1D_CTRL Reordering State Output Table

Output	Value	State
DONE	'1'	S9
	'0'	All other states
INC_ADR_CTL	'1'	S4
	'0'	All other states
REORDER_ADR_CTL	"01"	S6
	"10"	S2, S8
	"00"	All other states
REORDER_STATUS_CTL	'1'	S1, S7
	'0'	All other states
SHIFT_MODE_out	"10"	S2, S4
	"00"	All other states
RW_out	"01"	S5, S7
	"10"	S1, S3
	"00"	All other states

3.2.8. SLIDING_WINDOW

The SLIDING_WINDOW module includes a pixel storage buffer and arithmetic logic that is primarily used to perform the lifting scheme and reordering.

The CDF 5/3 SLIDING_WINDOW unit consists of a four-pixel buffer along with a LIFT_ALU_GROUP consisting of two LIFT_ALU units. Figure 3.14 shows the SLIDING_WINDOW unit forward DWT configuration for the CDF 5/3 wavelet.

Figure 3.14: Forward Configuration of SLIDING_WINDOW unit for CDF 5/3 DWT

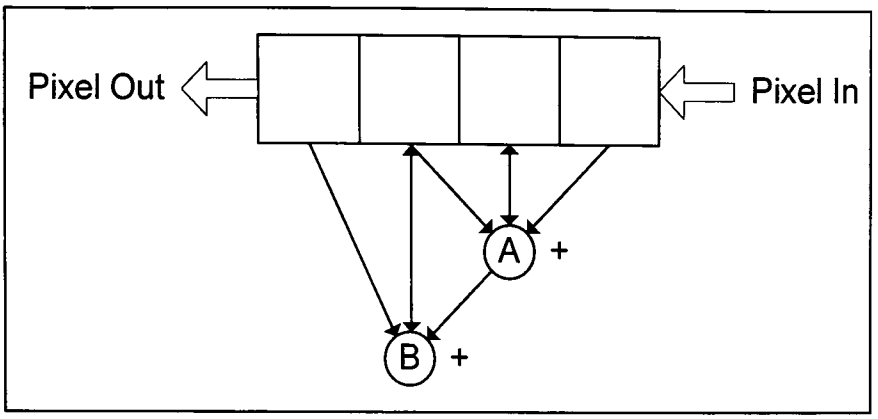
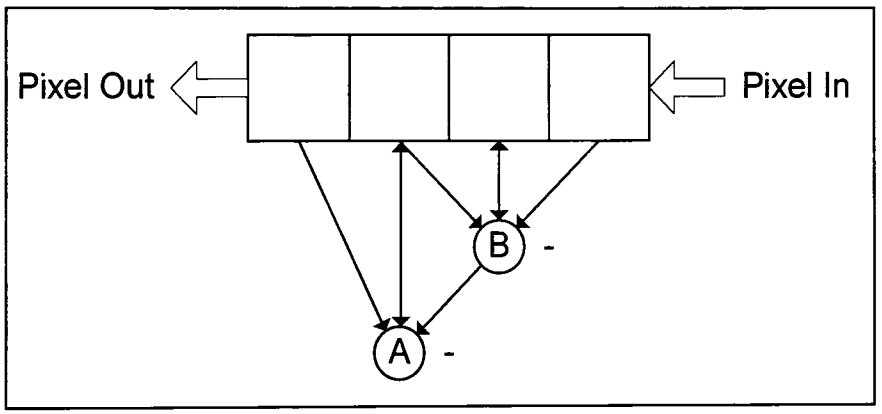


Figure 3.15 shows the SLIDING_WINDOW unit inverse DWT configuration for the CDF 5/3 wavelet.

Figure 3.15: Inverse Configuration of SLIDING_WINDOW unit for CDF 5/3 DWT



The CDF 9/7 SLIDING_WINDOW unit consists of a six-pixel buffer along with a LIFT_ALU_GROUP consisting of four LIFT_ALU units and two SCALU_ALU units. Figure 3.16 shows the SLIDING_WINDOW unit forward DWT configuration for the CDF 9/7 wavelet.

Figure 3.17 shows the SLIDING_WINDOW unit inverse DWT configuration for the CDF 9/7 wavelet. A separate set of SCALU_ALUs that are the inverse of the forward DWT SCALU_ALUs are used for the inverse DWT. If the SCALU_ALUs are inverses of each other then a second set of SCALE_ALUs need not be used and the SCALE_ALUs from the forward DWT are reused for the inverse DWT.

Figure 3.16: Forward Configuration of SLIDING_WINDOW unit for CDF 9/7 DWT

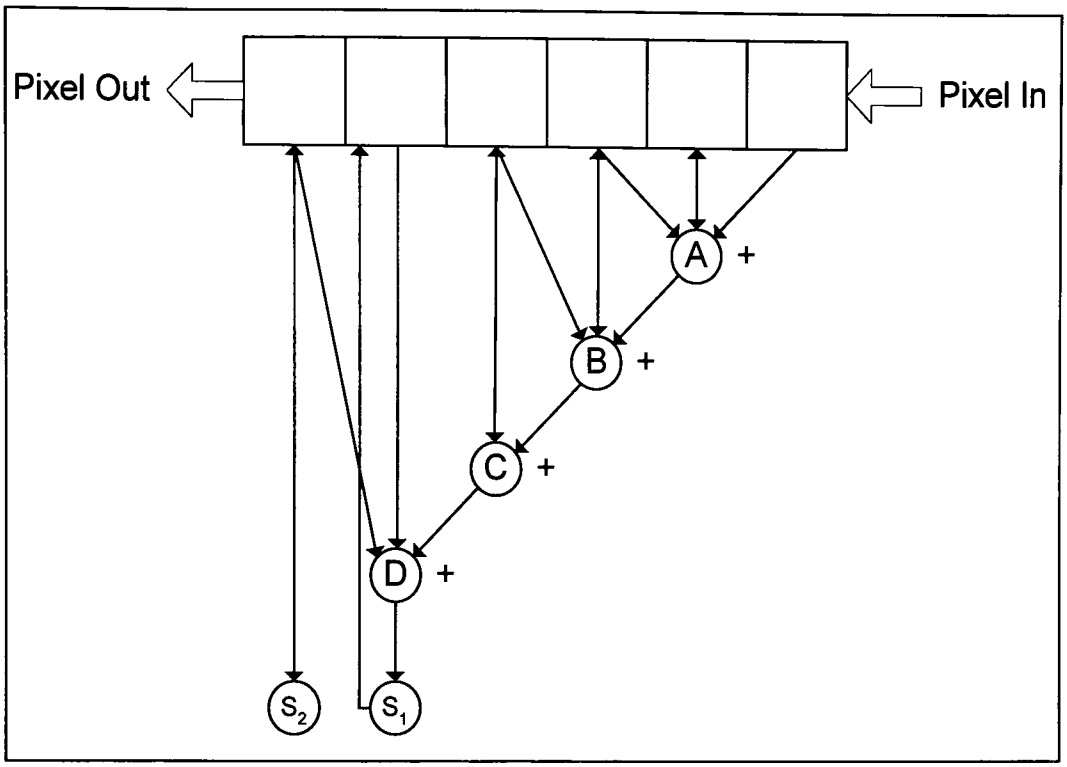
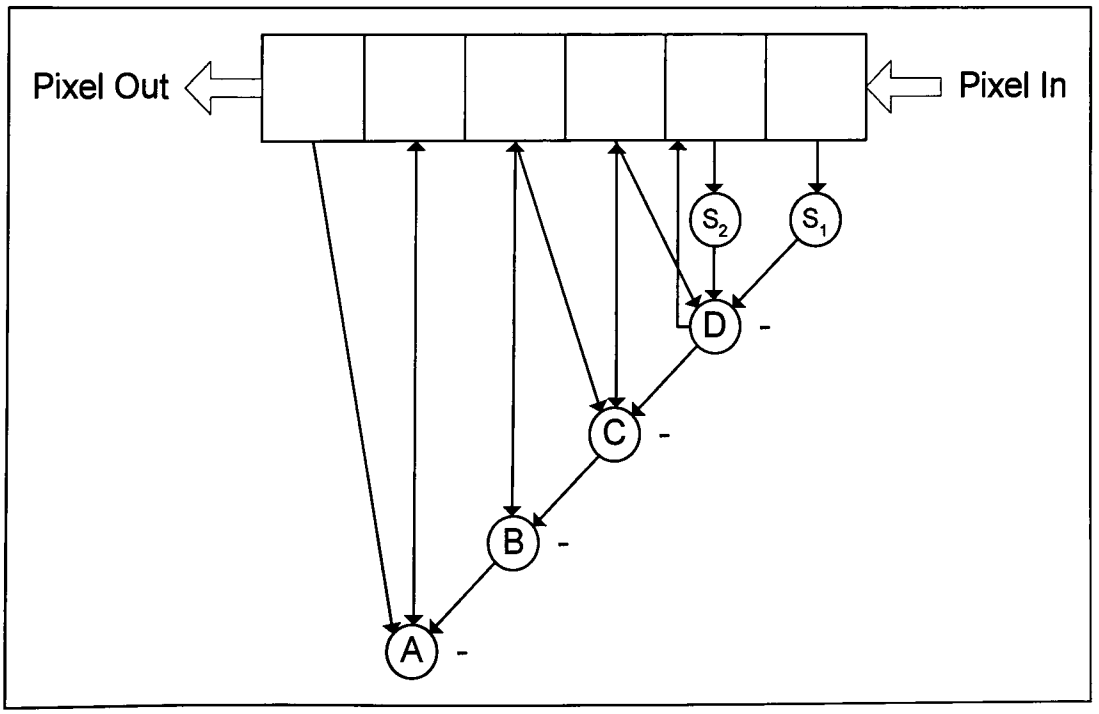


Figure 3.17: Inverse Configuration of SLIDING_WINDOW unit for CDF 9/7 DWT



3.2.9. LIFT_ALU_GROUP

The LIFT_ALU_GROUP unit contains all the individual lift and scale units required to perform the lifting scheme. The lift and scale units are connected together to perform either the forward or inverse DWT. The lift_alu_group generates the necessary hardware based on an array of lift_alu units and scale_alu units defined in the wavelet_pkg.vhd file.

3.2.10. LIFT_ALU

The LIFT_ALU unit is an ALU that performs a lifting step operation on a pixel (C) given its neighboring left (A) and neighboring right (B) pixels, lifting step coefficient (K) and, if needed, an extra constant (L). The lifting step coefficient (K) can be any positive/negative real number. A floor, ceiling, round, or truncate function is applied to the result of the multiplication to keep an integer value for the integer wavelet transform. The format for the lifting step is shown below.

$$C_{new} = C + [K(A + B + L)]$$

The lifting step equations for the CDF 5/3 and CDF 9/7 biorthogonal wavelets have L values of 0 and use the floor function as shown below.

$$C_{new} = C + \lfloor K(A + B) \rfloor$$

The lift unit has to ability to perform symmetric extension near the border of any image. For example if pixel C is on the leftmost border of the image there is only the neighboring right pixel (B) and no neighboring left pixel (A). Therefore using the symmetric extension method pixel B is used in place of pixel A and the lifting equation becomes the following.

$$C_{new} = C + \lfloor K(B + B) \rfloor = C + \lfloor K(2B) \rfloor$$

3.2.11. SCALE_ALU

The SCALE_ALU unit is an ALU that performs a scaling step operation on a pixel (A) given a scaling step coefficient (K). The scaling step coefficient (K) can be any positive/negative real number. A floor, ceiling, round, or truncate function is applied to the result of the multiplication to keep an integer value for the integer wavelet transform. The format for the scaling step is shown below.

$$A_{new} = \lceil K(A) \rceil$$

The scaling step equation for the CDF 9/7 biorthogonal wavelet uses the floor function as shown below.

$$A_{new} = \lfloor K(A) \rfloor$$

The CDF 5/3 biorthogonal wavelet does not require scaling steps.

3.2.12. FPT_COEF_MULT

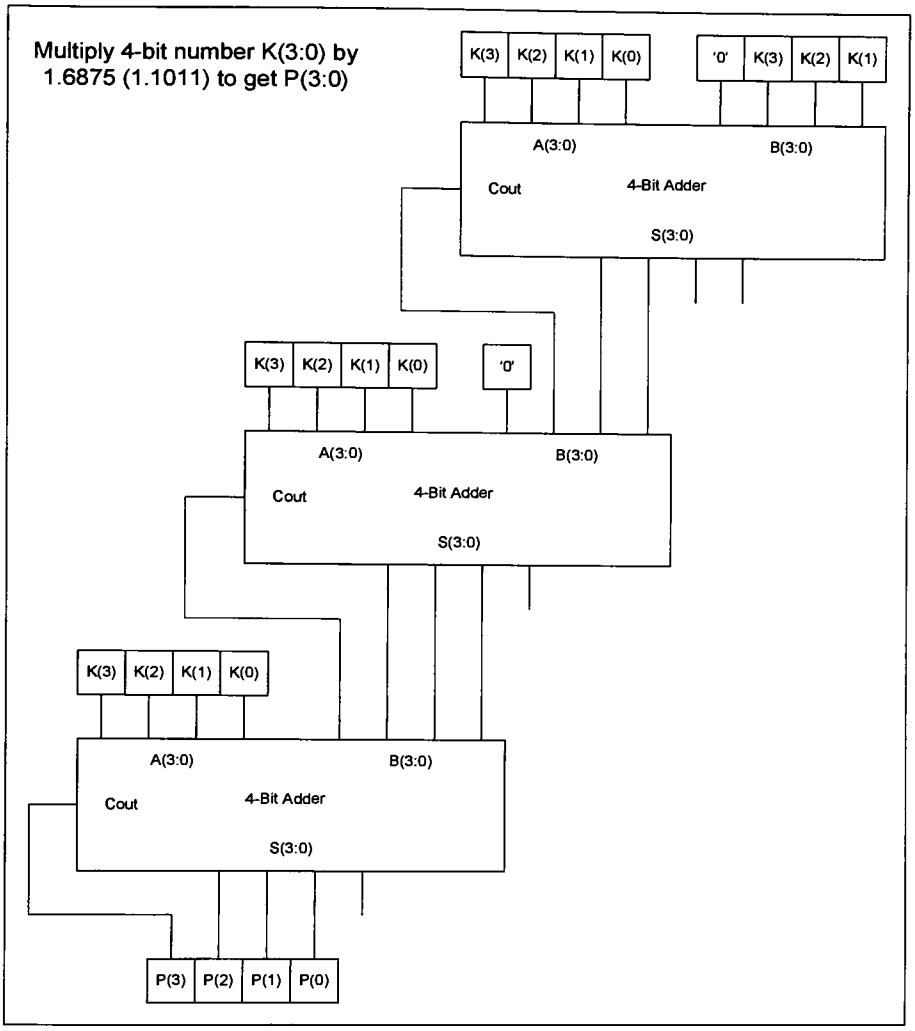
The FPT_COEF_MULT unit is a multiplier that performs fixed-point multiplication of an N-bit signed/unsigned integer number by a constant positive/negative real number. For synthesis purposes the generic value representing the real number coefficient is an array of integers indicating the location of each '1' in the binary number. The least significant '1' is indicated as 1, the bit to the left is 2, etc. Another integer generic is used to indicate the location of the decimal point, 1 meaning there is one decimal place. For example the number 1.6875 or binary 1.1011 is formatted as follows:

coefficient array: {1,2,4,5}

decimal location: 4

The generated hardware is shown in Figure 3.18.

Figure 3.18: Example of Generated FPT_COEF_MULT Hardware



3.3. DWT Core Implementation

The DWT core is modeled in various stages before final implementation on an FPGA. A high-level MATLAB description is used to prove correct functionality of the DWT, a VHDL description is used to provide a low-level hardware description of the DWT core to be used for simulation and eventually for synthesis of the hardware DWT core onto an FPGA.

3.3.1. MATLAB Implementation

The DWT core is modeled using the MATLAB language. The wide range of image processing tools and matrix manipulation operators provided by MATLAB make it very suitable for this application.

The MATLAB DWT core model is packaged as a MATLAB function that has analogous inputs and output to the hardware DWT core. The control lines of the hardware DWT core are provided to the MATLAB model as function parameters. Rather than reading/writing the image from/to a memory component the MATLAB model inputs/outputs the image as parameters. The MATLAB DWT core model serves only as functional verification for the DWT core. This model provides no hardware information such as timing, size, or power characteristics of the DWT core. The MATLAB DWT core model uses a software-optimized version of the DWT algorithm that is implemented in hardware DWT core. This model is optimized to exploit the efficient matrix manipulation operations of MATLAB to reduce simulation run times while producing the same results as the hardware DWT core. The MATLAB DWT core can be very easily modified providing maximum flexibility during the prototyping phase. Also this model can be easily integrated into testbenches. The MATLAB source code is found in Appendix C.

3.3.2. VHDL Implementation

The DWT core is modeled using the VHDL '93 hardware description language. The VHDL DWT core does not make use of any special constructs not available in VHDL '87 and therefore can be ported over easily. In addition the code can be ported over to Verilog with minimal effort.

The VHDL DWT core model serves two purposes. One purpose of this model is to serve as functional and timing verification of the DWT core. This model exactly mimics the operation of the actual hardware DWT core providing not only functional verification but timing information/verification as well. The second purpose of this model is to serve as synthesizable code for the target hardware platform providing all the necessary inputs and outputs.

The VHDL code is written at a low RTL level using data flow and structural descriptions with the exception of simple behavioral logic used for state machines. This allows the DWT to be synthesized with little effort and, if desired, aids in full-custom ASIC design using schematic capture. Arithmetic units are all custom-designed and do not have to be inferred at synthesis time. The DWT core makes extensive use of VHDL's built-in hardware generation capabilities such as generate and generic statements to tailor the hardware to the user-defined wavelet. The VHDL

DWT model cannot be as easily modified as the MATLAB model and takes much longer to simulate which makes it less desirable for prototyping. The VHDL source code is found in Appendix C.

3.3.3. FPGA Implementation

The VHDL DWT core model is synthesized and ported to an FPGA for hardware testing.

3.3.3.1. FPGA Prototyping Board

The XSV-300 FPGA prototyping board from Xess Corporation shown in Figures 3.19 and 3.20 is used for testing the hardware DWT core.

3.3.3.1.1. FPGA

The XSV-300 contains a 240-pin Xilinx XCV300 Virtex FPGA that is a 300,000 gate-equivalent FPGA using lookup tables.

3.3.3.1.2. Memory

The XSV-300 contains two independent banks of 512k x 16 SRAM for a total of 2 MB to be interfaced directly with the FPGA as shown in Figure 3.21. Each bank is composed of two Winbond AS7C4096 SRAMs. Each bank has a 16-bit data bus, a 19-bit address bus, and asynchronous active-low chip enable (ce), output enable (oe), and write enable (we).

3.3.3.1.3. I/O

The XSV-300 provides a Dallas D1075 programmable clock oscillator to adjust the clock speed (or an external clock can be used) (Figure 3.22), various input switches (Figure 3.23) and output LEDs (Figure 3.24), and an interface to PC for downloading/uploading images directly to/from the SRAM memory.

Figure 3.19: Picture of XSV-300 FPGA Prototyping Board

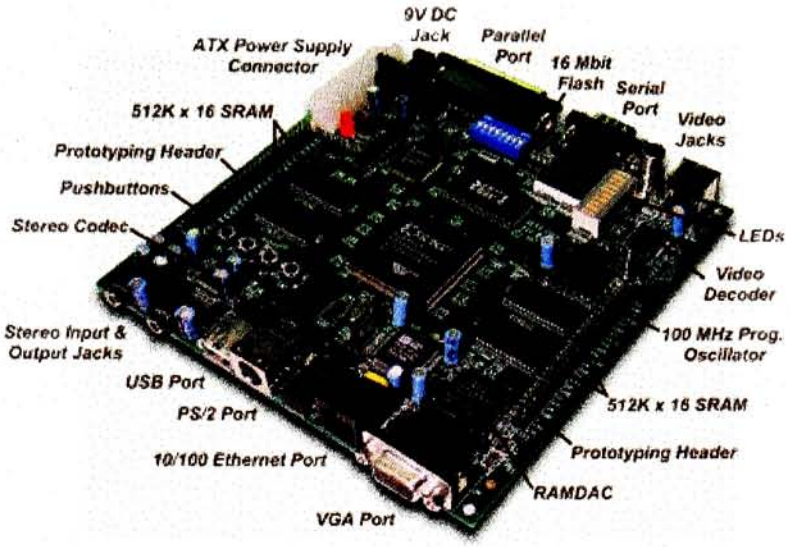


Figure 3.20: Module Diagram of XSV-300 FPGA Prototyping Board

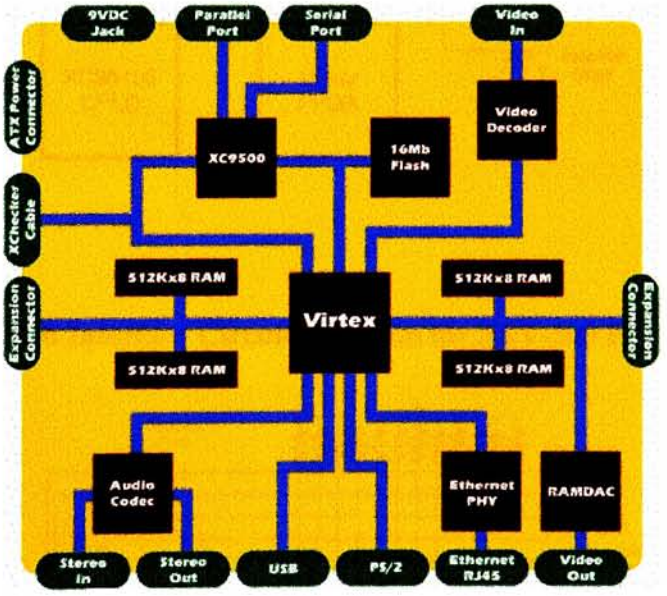


Figure 3.21: SRAM Interface Circuit Diagram for XSV-300 FPGA Prototyping Board

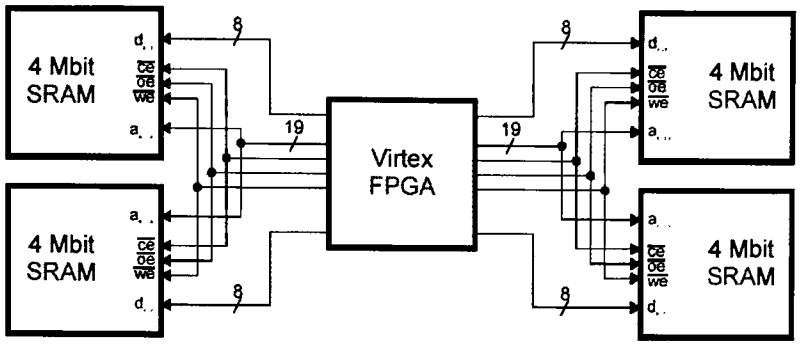


Figure 3.22: Clock Interface Circuit Diagram for XSV-300 FPGA Prototyping Board

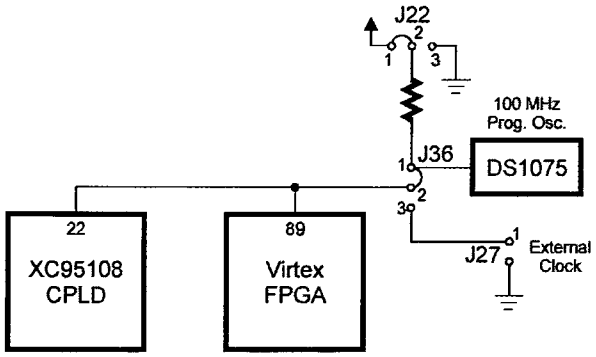


Figure 3.23: Input Switch Interface Circuit Diagram for XSV-300 FPGA Prototyping Board

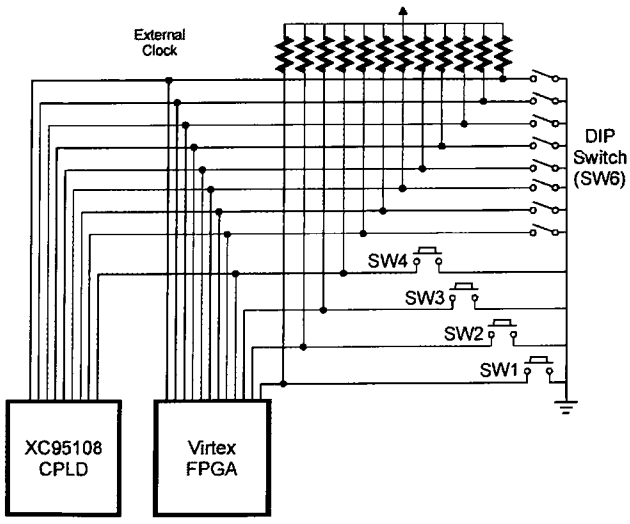
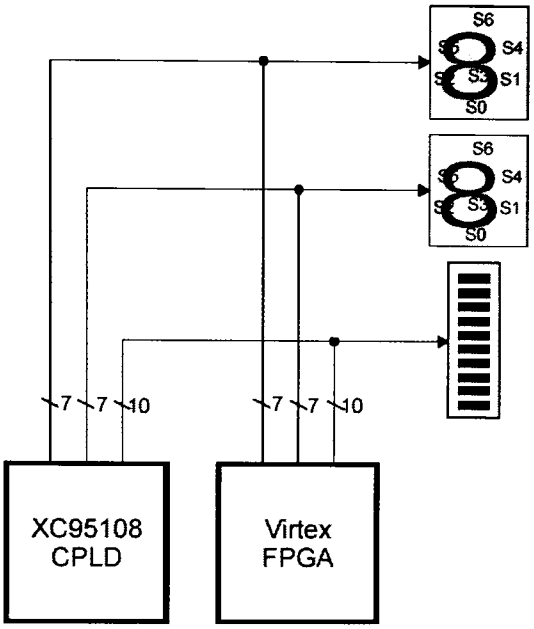


Figure 3.24: Output LED Interface Circuit Diagram for XSV-300 FPGA Prototyping Board



3.3.3.2. Synthesis

The VHDL DWT core is synthesized to the XSV-300 board using the Xilinx ISE 4.2 software. The ports on the DWT core are mapped to the corresponding memory data bus, address bus, and control lines as well as input switches and output LEDs using the Xilinx configuration utility. The clock port is connected to the FPGA system clock pin.

Chapter 4. Results

This chapter discusses the verification of the DWT core. A performance analysis of the DWT core is given along with results from hardware synthesis. Finally the results from applying wavelet denoising to images are discussed.

4.1. DWT Core Verification

Verification is performed on the DWT core to ensure the 2-D DWT is correctly performed on an image. First the MATLAB model of the DWT core is verified against the traditional convolution-based DWT to prove that the DWT is being performed correctly. Once the MATLAB model is proven correct the VHDL model is verified against it. The VHDL model is shown to produce the same output as the MATLAB model and therefore performs the DWT correctly. Finally the DWT core is implemented in hardware. The hardware implementation is shown to produce the same output as both the VHDL and MATLAB models and therefore performs the DWT correctly.

Both the CDF 5/3 and CDF 9/7 biorthogonal wavelets were implemented and verified. In all cases an 8-bit grayscale image is used. As mentioned earlier, an 8-bit sample requires four extra guard bits for a minimum of 8 + 4 = 12 bit scaling and wavelet coefficient representation for the CDF 5/3 DWT. For the CDF 9/7 DWT one extra guard bit is required for a minimum of 8 + 1 = 9 bit scaling and wavelet coefficient representation. Taking this into consideration, the fact that MATLAB provides a 16-bit signed data type and the SRAM onboard the FPGA prototyping board is 16-bits wide the scaling and wavelet coefficients, and therefore the DWT core datapath, is chosen to be 16-bits wide to avoid the overflow or underflow problems.

4.1.1. Testing Benchmarks

A 512 pixel by 512 pixel 8-bit grayscale version of the popular Lena (Lenna) image is used for MATLAB verification. This image has both contains many different features and textures making it a good random test stimulus. The Lena image is shown in Figure 4.1.

Figure 4.1: Lena 512x512 Pixel 8-bit Grayscale Image

Since wavelet coefficients are generally very small compared to the scaling coefficients, they are difficult to observe visually. Histogram equalization is applied to each of the wavelet subbands of a transformed image to more clearly illustrate them. By doing so the larger wavelet coefficients are more easily visualized in the image. In addition, the absolute value of the wavelet coefficients are used since images cannot contain negative pixel values.

Two metrics are used in this thesis to measure the difference between two images. The first measure is Peak Signal-To-Noise Ratio (PSNR). PSNR is defined as the equation below.

$$\text{PSNR} = 20 \bullet \log_{10}\left(\frac{K}{\text{RMS}}\right)$$

K is the largest possible value of the signal (255 in this case). RMS is the root mean square difference between the two images. PSNR is given in decibels (dB) that measure the ratio of the peak signal and the difference between two images. An increase of 20 dB corresponds to a ten-fold decrease in the RMS difference between the two images. [33 (chapter 2)]

The second measure is custom-designed for this thesis and is called the percent difference. Percent Difference is defined as the equation below.

$$\text{Percent Difference} = \frac{\sum_{y=0}^{(n\_rows-1)} \sum_{x=0}^{(n\_cols-1)} \text{abs}(\text{imageA}[x,y] - \text{imageB}[x,y])}{(n\_cols \bullet n\_rows)(K+1)}$$

K is the largest possible value of the signal (255 in this case), n_cols is the number of columns in the image (image width) and n_rows is the number of rows in the image (image height). The percent difference seeks to determine the percentage that one image differs from another image on a scale from 0 being the least similar to 1 being identical. The percent difference is computed by computing the absolute value differences between each pixel in an image and their corresponding pixels in the other image. These differences are then added together and divided into the total maximum intensity of an image.

The PSNR and percent difference metrics typically yield the same results. Two metrics are used since occasionally one metric incorrectly identifies two images as being very similar/dissimilar when in fact visually they are not.

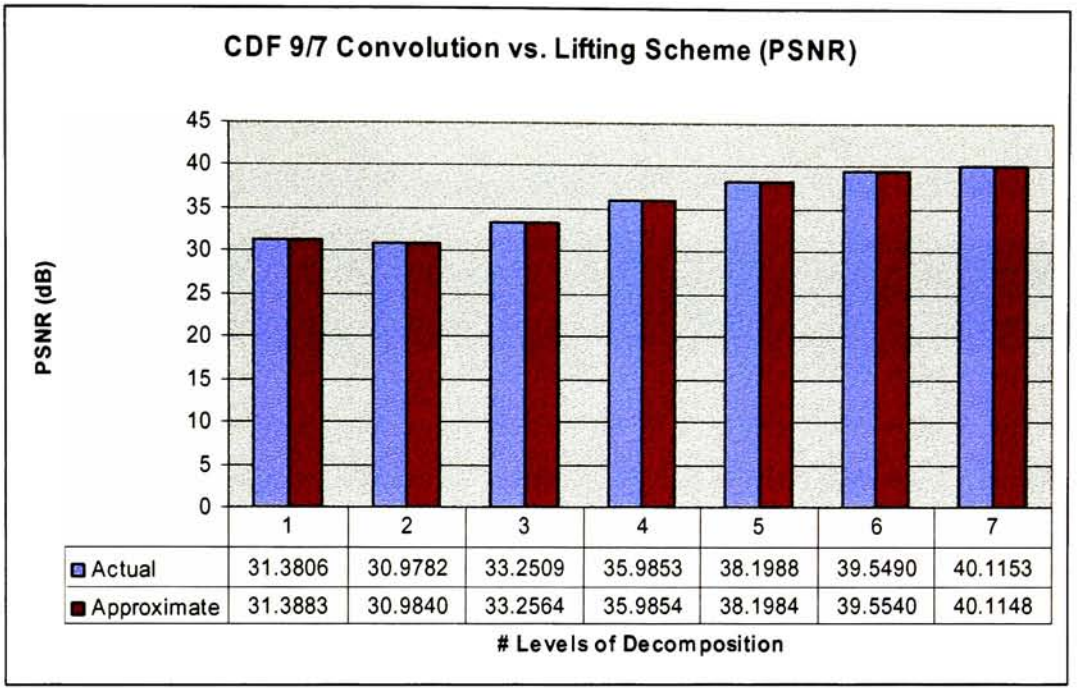
4.1.1. MATLAB Verification

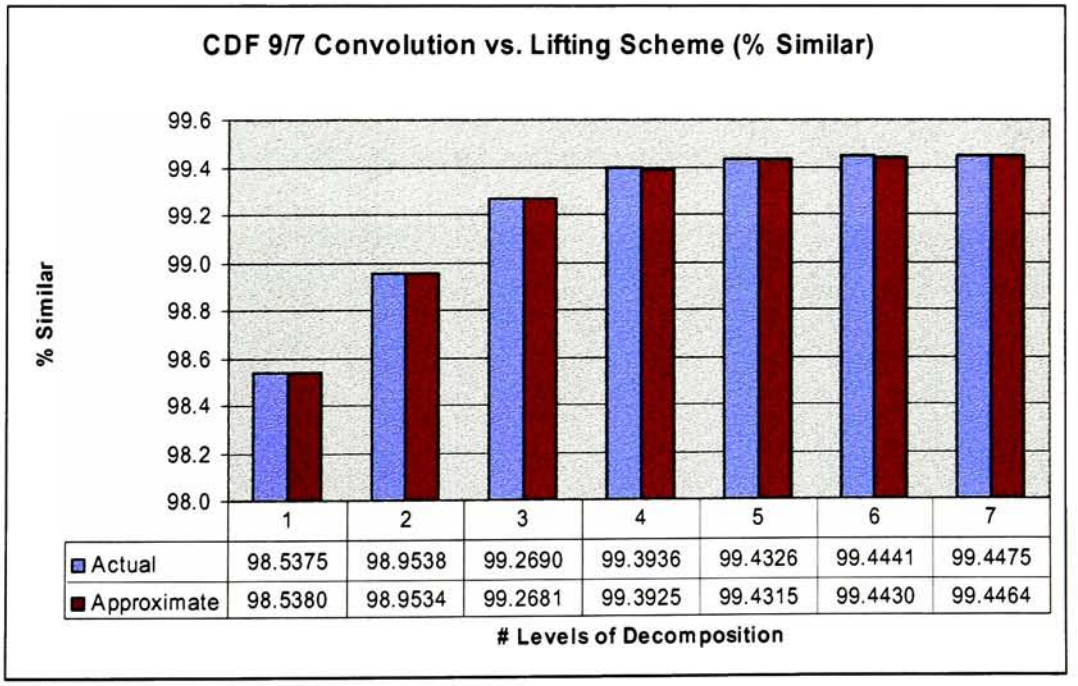
The MATLAB implementation of the DWT core is verified against the standard convolution-based DWT to prove correct functionality of the DWT core. MATLAB 6.x software from Mathworks, Inc. is used for simulation of the MATLAB model. The MATLAB model is in the form of a MATLAB function that inputs a .tif image along with a set of user commands to produce an output .tif image.

The traditional convolution-based CDF 5/3 DWT is applied to the Lena image for 1 to 8 levels of decomposition. Next the MATLAB model of the DWT core using the CDF 5/3 wavelet is applied to the Lena image for 1 to 9 levels of decomposition. The resulting images are found in Appendix A and were identical as expected.

The traditional convolution-based CDF 9/7 DWT is applied to the Lena image for 1 to 7 levels of decomposition. Next the MATLAB model of the DWT core using both the actual full precision lifting coefficients and the approximated lifting coefficients for the CDF 9/7 wavelet are applied to the Lena image for 1 to 9 levels of decomposition. The resulting images are found in Appendix A. Figure 4.2 shows the PSNR and percent difference measurements.

Figure 4.2: Comparison of Lena Image Transformed using CDF 9/7 DWT Convolution and Lifting Scheme for Multiple Levels of Decomposition





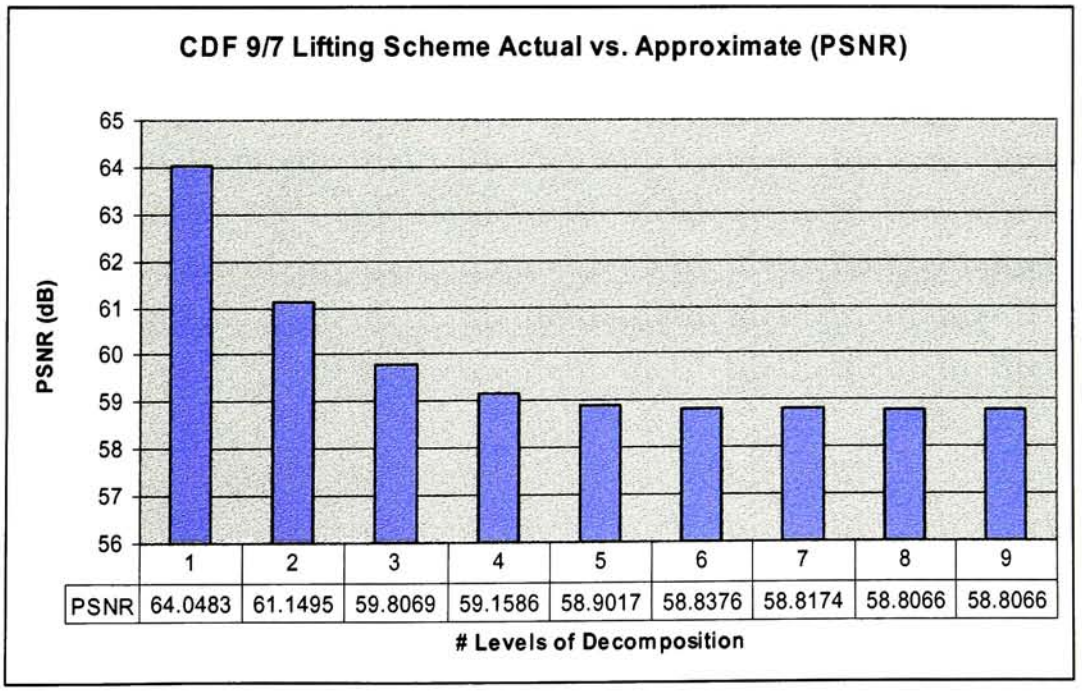
The reason for deviation between the convolution and lifting scheme DWT images is due to the fact that the scaling factors used are from the JPEG2000 standard that are different from the scaling factors derived directly from the filters, however, this did not have a major impact.

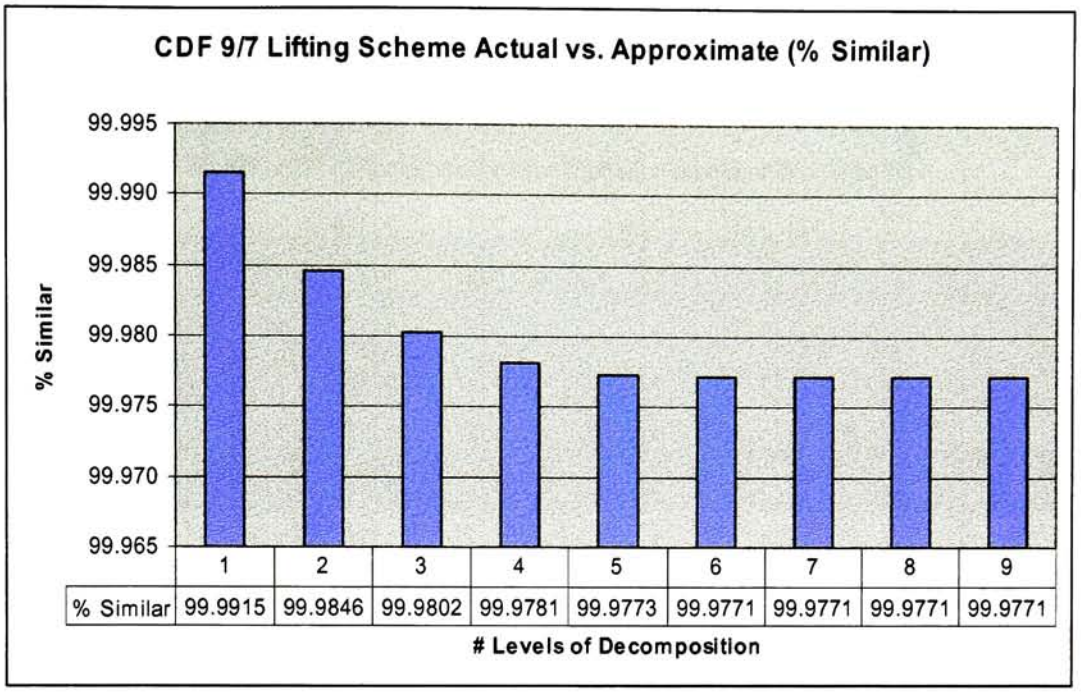
The actual and approximate coefficients that are used are shown in the table 4.1. The goal was to keep the difference between the actual and approximated coefficients all below 0.0001. This threshold was arbitrarily chosen based on desired hardware vs. precision constraints and is not intended to be the optimal choice.

Table 4.1: CDF 9/7 Actual vs. Approximated Coefficient Values and Error

Step	Actual	Approximate (decimal)	Approximate (binary)	Error
α	-1.586134342	-1.586181641	-1.100101100001	0.000047299
β	-0.05298011854	-0.052978516	-0.000011011001	0.000001602
γ	0.8829110762	0.882873535	0.11100010000001	0.000037540
δ	0.4435068522	0.443542480	0.011100011	0.000035628
ζ_1	1.149604398	1.149658203	1.001001100101	0.000053805
ζ_2	0.8698644523	0.8698730469	0.110111101011	0.0000085946

Figure 4.3: Comparison of Lena Image Transformed using CDF 9/7 DWT Lifting Scheme with Actual and Approximated Coefficients for Multiple Levels of Decomposition

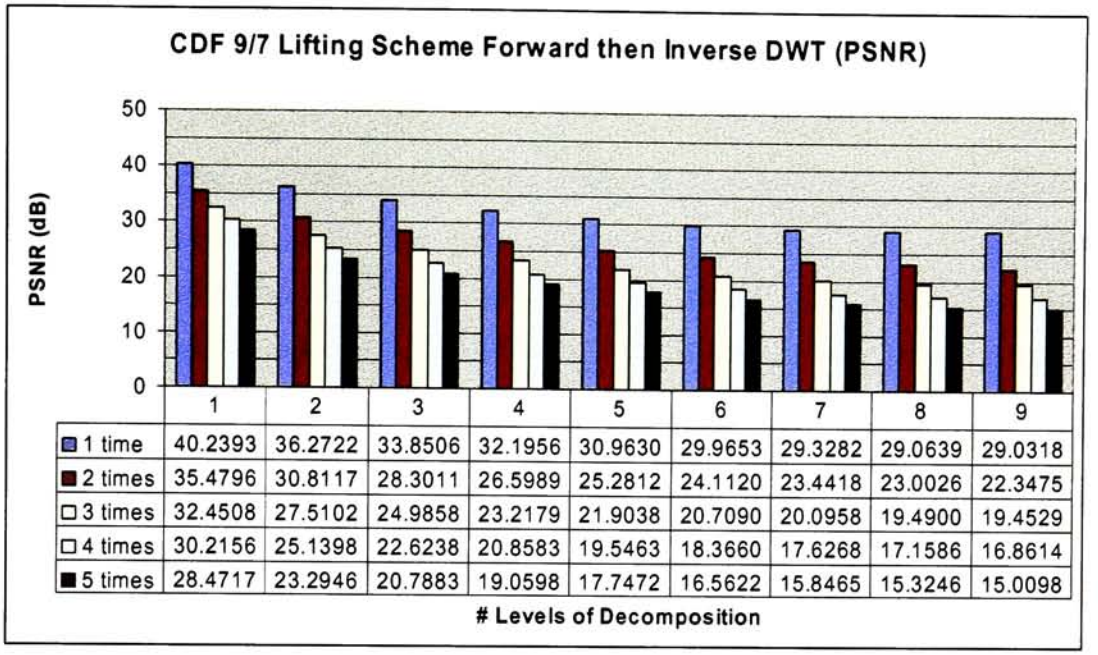




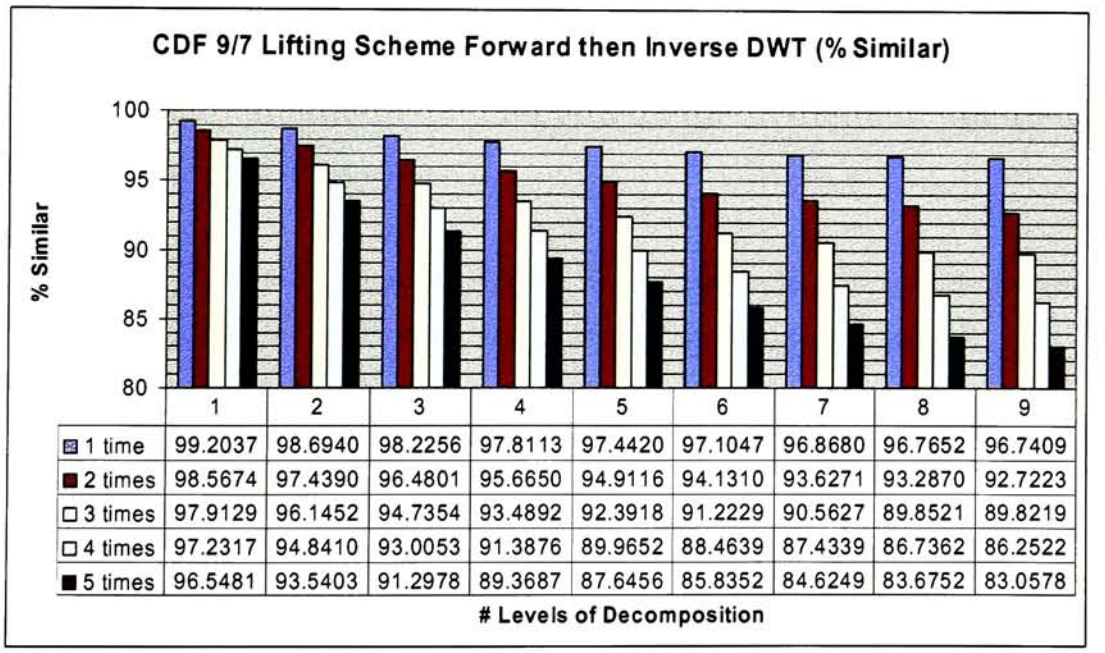
Visual inspection of the CDF 5/3 and the CDF 9/7 transformed images suggests that the CDF 9/7 wavelet provides better energy compaction having smaller wavelet coefficients as is expected.

To determine the reversibility of the DWT core the forward DWT is performed on an image followed by the inverse DWT. This is performed for 1 to 9 levels of decomposition of the image and is repeated from 1 to 5 times to show the degradation of an image as multiple transforms are applied to it. The CDF 5/3 DWT is fully reversible. For every level of decomposition the CDF 5/3 DWT demonstrated perfect reconstruction with the transformed image being identical to the original. The CDF 9/7 DWT on the other hand is not fully reversible due to finite-precision effects. The reconstructed image demonstrates higher levels of degradation at higher levels of decomposition and the more times the forward and inverse transform are performed.

Figure 4.4: Comparison of Lena Image Transformed and Reconstructed Multiple Times using Lifting Scheme CDF 9/7 DWT for Multiple Levels of Decomposition



	1	2	3	4	5	6	7	8	9
1 time	40.2393	36.2722	33.8506	32.1956	30.9630	29.9653	29.3282	29.0639	29.0318
2 times	35.4796	30.8117	28.3011	26.5989	25.2812	24.1120	23.4418	23.0026	22.3475
3 times	32.4508	27.5102	24.9858	23.2179	21.9038	20.7090	20.0958	19.4900	19.4529
4 times	30.2156	25.1398	22.6238	20.8583	19.5463	18.3660	17.6268	17.1586	16.8614
5 times	28.4717	23.2946	20.7883	19.0598	17.7472	16.5622	15.8465	15.3246	15.0098



	1	2	3	4	5	6	7	8	9
1 time	99.2037	98.6940	98.2256	97.8113	97.4420	97.1047	96.8680	96.7652	96.7409
2 times	98.5674	97.4390	96.4801	95.6650	94.9116	94.1310	93.6271	93.2870	92.7223
3 times	97.9129	96.1452	94.7354	93.4892	92.3918	91.2229	90.5627	89.8521	89.8219
4 times	97.2317	94.8410	93.0053	91.3876	89.9652	88.4639	87.4339	86.7362	86.2522
5 times	96.5481	93.5403	91.2978	89.3687	87.6456	85.8352	84.6249	83.6752	83.0578

When only one forward and inverse transformation is applied the reconstructed image is almost visually identical to the original image as shown in Appendix A. At all levels of decomposition the images show noticeable degradation as the forward and inverse transforms are applied a number of times.

4.1.2. VHDL Verification

The VHDL model of the DWT core is verified against the MATLAB model for correct functionality. ModelSim from Model Technology is the VHDL model simulator used to test the VHDL code.

In order to test the functionality of the VHDL model it must be placed within a testbench to simulate a working environment. The testbench provides the DWT core with all the required stimulus such as user inputs to the core and an SRAM memory model to interface with. The SRAM memory model is coded in behavioral VHDL and is found in the "sram_model.vhd" file in the appendix. The testbench connecting the DWT core to the SRAM memory model is found in the "sim_test.vhd" file in the appendix.

The UNIX picture viewing program X-View (XV) is used to convert any type of image to a portable grayscale pixel map (.pgm) image file format. The .pgm format represents an 8-bit grayscale image as raw data in the form of ASCII characters. The SRAM model downloads a .pgm image into its memory contents at initialization and uploads the memory contents to another .pgm file when the DWT core has finished operation. The resulting .pgm file can then be converted back to any file format by XV.

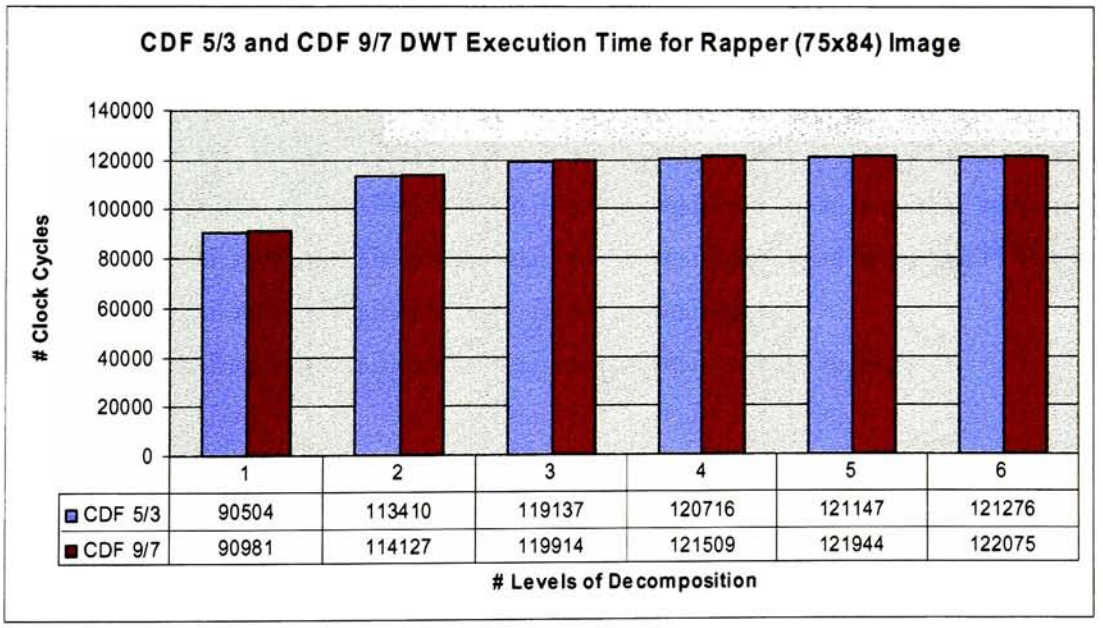
Due to the memory limitations of the computers used to simulate the VHDL model images larger than 128x128 pixels are not feasible to simulate and in some cases do not run at all.

A 75 pixel by 84 pixel image called "rapper" is used to verify the VHDL model shown in Figure 4.5.

Figure 4.5: Rapper 75x84 Pixel 8-bit Grayscale Image

Both the CDF 5/3 and CDF 9/7 forward and inverse DWTs are applied for 1 to 6 levels of decomposition using a clock period of 10 ns. The results are shown in Figure 4.6.

Figure 4.6: Execution Time for CDF 5/3 and CDF 9/7 DWT on Rapper Image for Multiple Levels of Decomposition



As expected the number of clock cycles required to perform an addition level of decomposition is approximately 25% more than the next level down. The execution time of the CDF 9/7 is slightly larger due to the extra clock cycles required to fill and empty the larger sliding window buffer.

It is determined from these results that an average of approximately 14.5 clock cycles per pixel are required to perform the DWT. Eight clock cycles are required for the necessary memory read and writes while the remaining six clock cycles are used for control and arithmetic.

The results of the VHDL DWT core operations on the rapper image are shown in Appendix A. The VHDL results are found to be identical to the MATLAB results and therefore the VHDL model is correct.

4.1.3. FPGA Hardware Verification

The final step in verifying the DWT core is the hardware implementation. The verification of the hardware implementation of the DWT core is performed using the on the FPGA prototyping board. A testbench similar to the VHDL model containing the user inputs and an SRAM module is required. Since there are not enough input switches on the FGPA board for all the DWT core user inputs many of the inputs are hardcoded in the testbench at synthesis time. Certain inputs such as start, reset, done, forward/inverse transform, and number of levels of decomposition are connected directly to switches/LEDs. The memory control lines of the DWT core are directly connected to the FPGA board's SRAM. The "fgpa_test.vhd" file in the appendix is the hardware testbench. As with the VHDL simulation the XV program is used to convert and image to and from the pgm file format. A command line program written in C code is used to convert a .pgm image to a 16-bit xes memory file format suitable to download directly to the FPGA board's SRAM. This program, called "pgm2xes," has the following usage:

pgm2xes <input.pgm> <output.xes> <image width> <image height>

This first command line argument <input.pgm> specifies the pgm image file to be converted. The second command line argument <output.xes> specifies the name of the new 16-bit xes memory file that is to be generated. The third and fourth command line arguments <image width> and <image height> specify the width and height of the image. The "gxsload" program provided from Xess Corporation is used to download the xes file containing the image directly into the FGPA prototyping board's SRAM. Next the user provides the necessary input stimulus via the pushbuttons and DIP switches to perform the forward/inverse DWT on the image in the memory. Next the FPGA board's SRAM is uploaded to a xes memory file again using the

"gxsload" program. Finally another command line program written in C code is used to convert the uploaded .xes memory file back to the pgm image file format for viewing. This program, called "xes2pgm", has the following usage:

xes2pgm <input.xes> <output.pgm> <image width> <image height>

The source code for both of these programs is found in Appendix D.

Due to the complex self-generating hardware nature of the VHDL code for the DWT core the basic Xilinx FPGA synthesis tool encounters an unknown error during technology mapping. This problem is addressed by performing some "manual synthesis" meaning that the user manually generates statements and unrolls loops that a more sophisticated synthesis tool should be able to do automatically. Synthesis tools are able to generate hardware in one or two dimensions such as generic multipliers, but the VHDL core generates hardware in three dimensions such as chains of array multipliers and therefore certain synthesis tools cannot accommodate this. A solution to this is to use a VHDL source code generator to dynamically generate the necessary VHDL code that can be more easily synthesized. The CDF 5/3 hardware is obtained by performing minimal manual synthesis. The CDF 9/7 hardware would take much too long to manually synthesize without error and therefore is not performed. Despite this limitation some hardware synthesis results were still obtained for the CDF 9/7 DWT to be compared with the CDF 5/3 DWT.

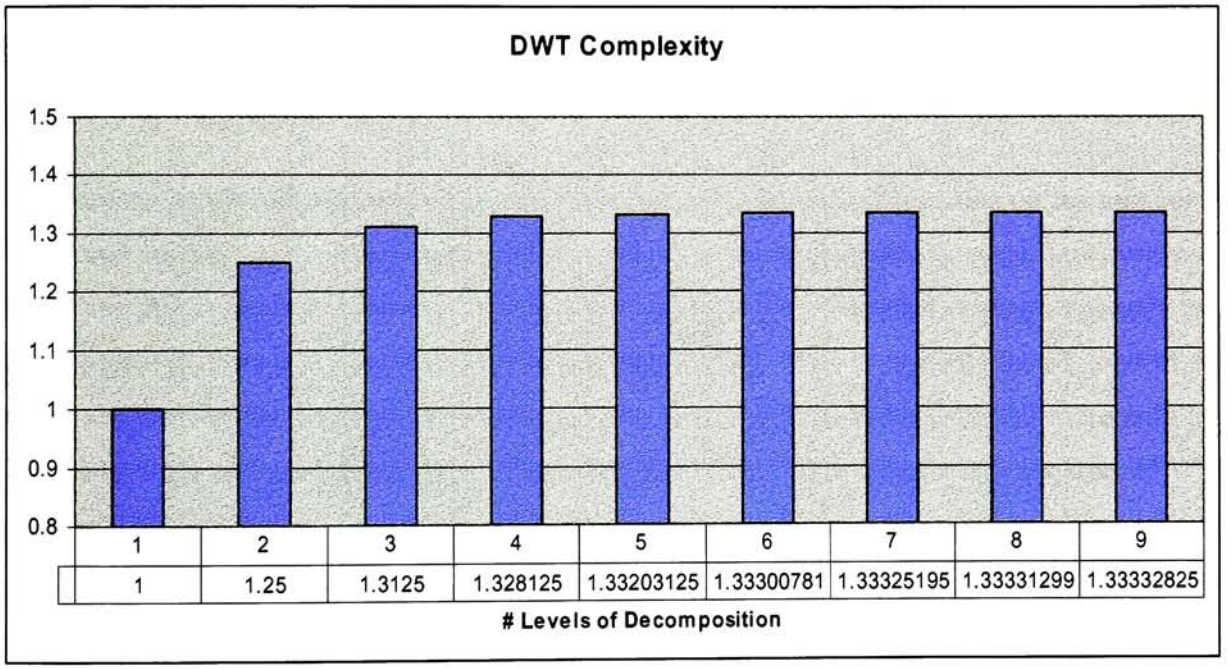
The maximum allowable size of an image to be manipulated by the DWT core is limited only by the amount of available SRAM. There is enough SRAM available on the board to process not only the small rapper image from the VHDL simulations but also the Lena 512x512 image from the MATLAB tests. Both images were processed with the rapper results identical to those of the VHDL and MATLAB simulations and the Lena results identical to the MATLAB simulation results proving the functionality of the hardware FPGA implementation. The VHDL design is not tailored specifically to Xilinx FPGAs and therefore should work on any other FPGA, ASIC or hardware technology.

4.2. Performance Analysis

4.2.1. DWT Complexity

Figure 4.7 illustrates the complexity of the DWT for different levels of decomposition, whether it is memory bandwidth or number of executions. Each successive level is 25% more complex than the previous level since only 25% the size of the previous level is transformed. The cost of performing additional levels of DWT reduces exponentially with the overall complexity approach, but never exceeding 1.333333... the complexity of performing the DWT for one level of decomposition.

Figure 4.7: Normalized Complexity of 2-D DWT for Different Levels of Decomposition



4.2.2. Memory Bandwidth

The memory bandwidth requirement of a hardware implementation is very important as this is typically the bottleneck in hardware applications. Memory reads and writes are often very costly and need to be minimized as much as possible. This section analyzes the memory bandwidth usage of different variations of the DWT such as convolution-based, lifting scheme-based and the DWT core design implemented for this thesis.

A detailed comparison of the memory bandwidth requirements is discussed for only the processing of the actual DWT and not for the optional reordering of the interleaved wavelet and

scaling coefficients since reordering is the same for each implementation. Each row of pixels would be read and then written back to with the reordered configuration. This is repeated for all rows of the image and then all the columns of the image. An N pixel by M pixel image requires 2*N*M reads and 2*N*M writes for a total of 4*N*M additional memory reads and writes for reordering.

All convolution-based DWTs have the same memory bandwidth regardless of size of the wavelet. The two high pass and low pass filters/convolution kemels each read all the pixels in a row and write the results to a buffer. This is repeated for all rows of the image and then all the columns of the image. Therefore, an N pixel by M pixel image requires 4*N*M reads and 4*N*M writes for a total of 8*N*M memory reads and writes for either the convolution-based CDF 5/3 or CDF 9/7 DWT.

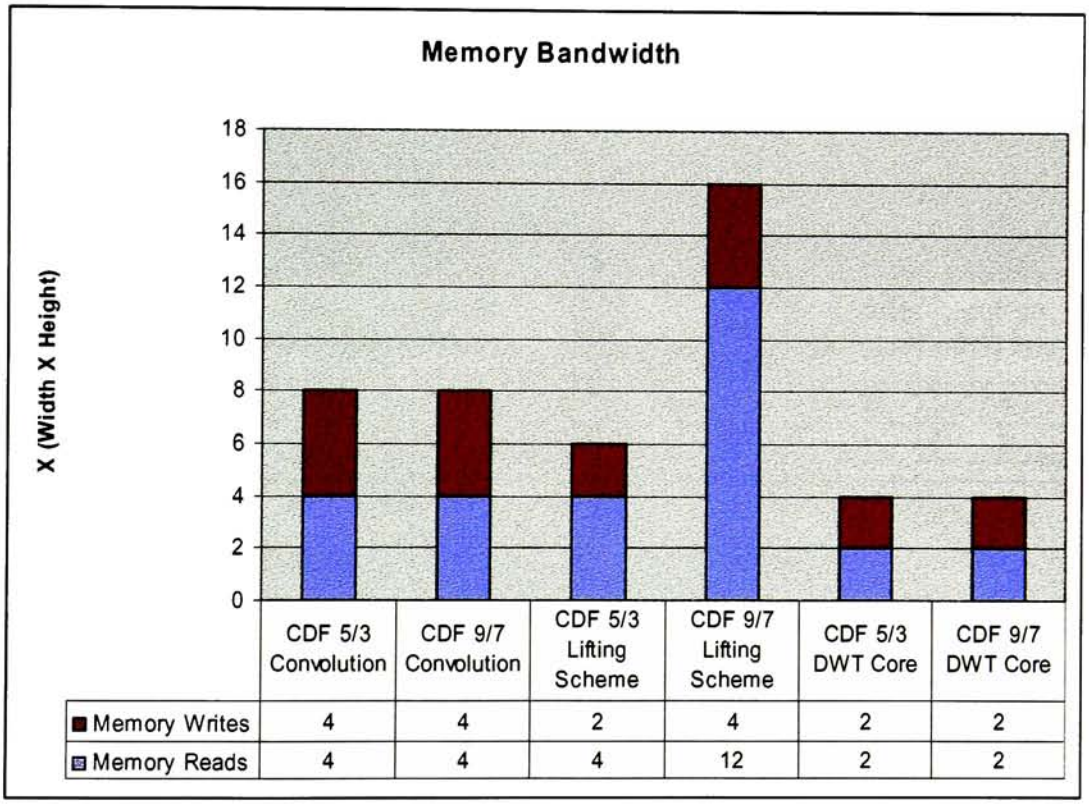
The memory bandwidth requirement using the lifting scheme varies depending on the number of lifting and scaling steps. Every pixel in a row is read once for each lifting stage, however, only half of the pixels are written back to memory each lifting stage. For each scaling stage only half of the pixels in a row are read from and written to memory. This is repeated for all rows of the image and then all the columns of the image. The lifting scheme based CDF 5/3 there only has two lifting steps and no scaling steps. Therefore, an N pixel by M pixel image requires 4*N*M reads and 2*N*M writes for a total of 6*N*M memory reads and writes for the lifting scheme based CDF 5/3 DWT. The lifting scheme based CDF 9/7 has four lifting steps and two scaling steps requiring a much higher memory bandwidth. An N pixel by M pixel image requires 12*N*M reads and 4*N*M writes for a total of 16*N*M memory reads and writes for the lifting scheme based CDF 9/7 DWT.

The memory bandwidth requirements for the DWT core designed for this thesis is lower than both the convolution-based and lifting scheme implementations. The DWT core reads each pixel is each row only once and writes each pixel back only once, regardless of the wavelet used. This is repeated for all rows of the image and then all the columns of the image. The lifting scheme based CDF 5/3 there only has two lifting steps and no scaling steps. Therefore, an N pixel by M

pixel image requires 2*N*M reads and 2*N*M writes for a total of 4*N*M memory reads and writes for either the DWT core implementation of the CDF 5/3 or CDF 9/7 DWT.

Figure 4.8 illustrates the memory bandwidth requirements in terms of memory reads and writes for performing the forward DWT on an image for 1 to 9 levels of decomposition using the different implementations of the DWT.

Figure 4.8: Normalized Memory Bandwidth (Reads/Writes) for CDF 5/3 and CDF 9/7 DWT using Convolution, Lifting Scheme, and DWT Core

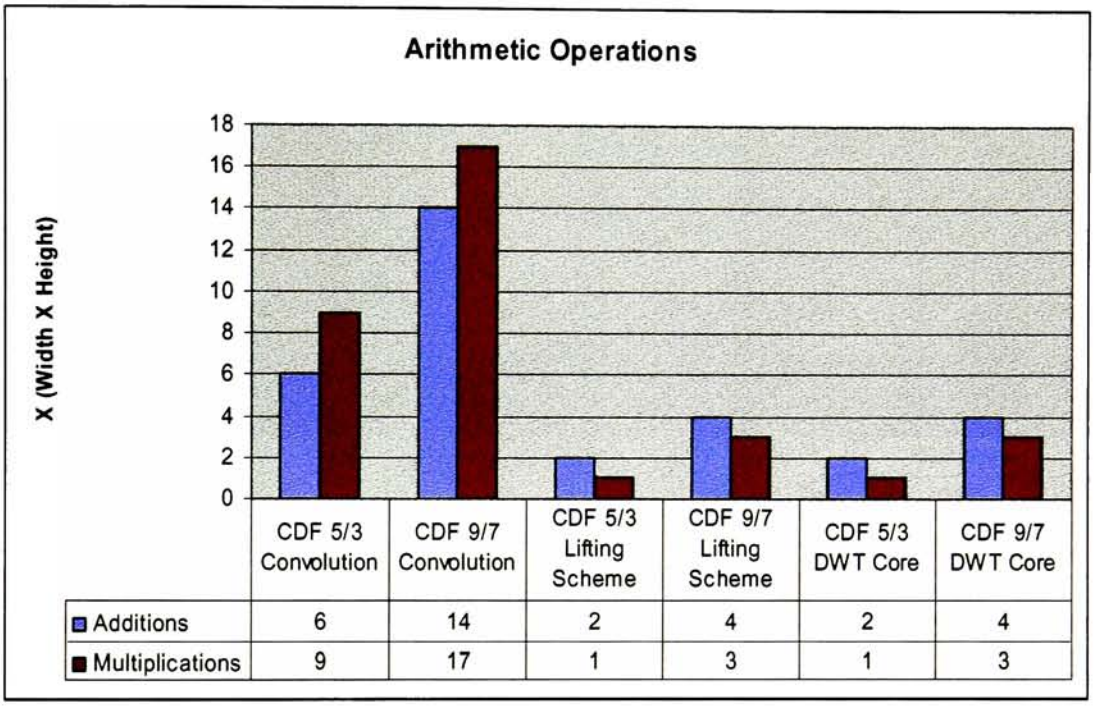


4.2.3. Arithmetic Operations

Figure 4.9 compares the arithmetic complexity of performing the CDF 5/3 and CDF 9/7 DWT using convolution, the lifting scheme, and the technique employed by the DWT core. Obviously the number of arithmetic operations performed for the CDF 9/7 are more than those of the CDF 5/3 for all implementations. The numbers of arithmetic operations for the lifting scheme and DWT core implementations are identical to each other, however, they are significantly less than convolution. Also the balance of addition and multiplication for the lifting scheme and DWT

core implementations is more desirable than convolution since a majority of the computation is performed using addition rather than multiplication.

Figure 4.9: Normalized Arithmetic Operations for CDF 5/3 and CDF 9/7 DWT using Convolution, Lifting Scheme, and DWT Core



	CDF 5/3 Convolution	CDF 9/7 Convolution	CDF 5/3 Lifting Scheme	CDF 9/7 Lifting Scheme	CDF 5/3 DWT Core	CDF 9/7 DWT Core
Additions	6	14	2	4	2	4
Multiplications	9	17	1	3	1	3

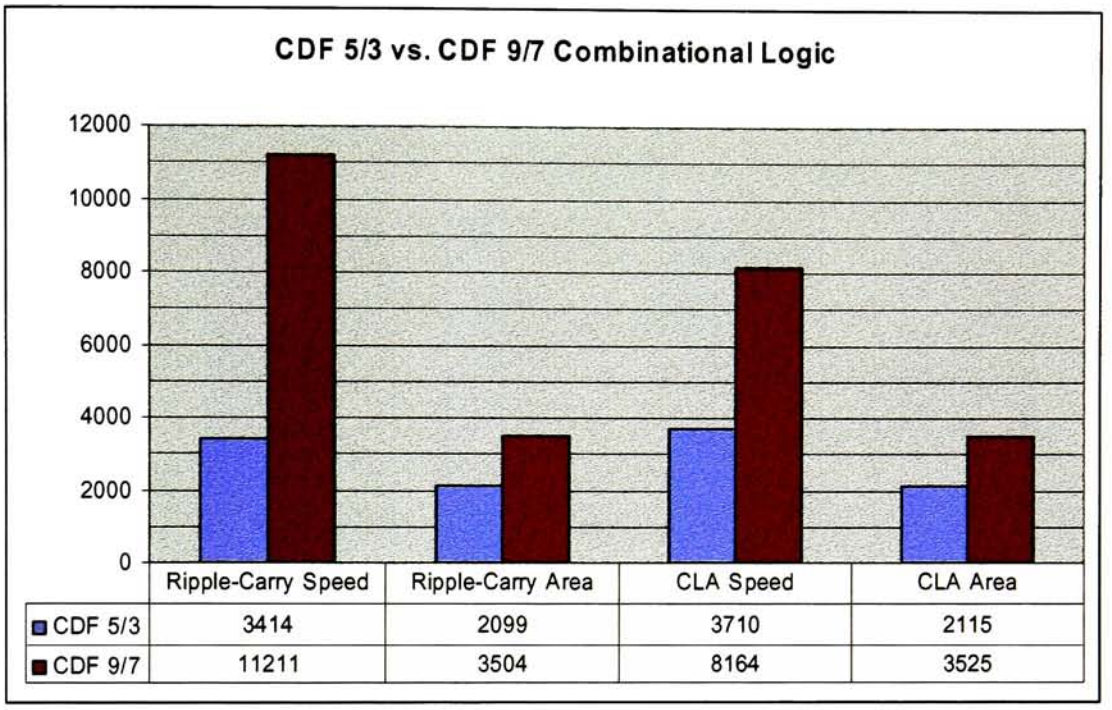
4.3. Synthesis Results

The DWT core is synthesized from VHDL to FPGA hardware using Xilinx ISE Foundation 4.2i software and the XST (Xilinx Synthesis Technology) synthesizer. The DWT core is synthesized for both CDF 5/3 and CDF 9/7, maximum speed and minimum hardware area, and Ripple-Carry and Carry-Look-Ahead (CLA) architectures.

4.3.1. Hardware Area

The amount of synthesized combinational logic obtained from the synthesis reports is shown in Figure 4.10. When synthesized for maximum speed the amount of combinational logic generated for the CDF 9/7 is more than double that for the CDF 5/3 DWT.

Figure 4.10: Combinational Logic Units for CDF 5/3 and CDF 9/7 DWT Synthesized for Maximum Speed/Minimum Area using Ripple-Carry/CLA adders



	Ripple-Carry Speed	Ripple-Carry Area	CLA Speed	CLA Area
CDF 5/3	3414	2099	3710	2115
CDF 9/7	11211	3504	8164	3525

The amount of synthesized sequential logic obtained from the synthesis reports is shown in Figure 4.11. In all cases the synthesized sequential logic for the CDF 9/7 is slightly more than that of the CDF 5/3 due to the similar control logic that is generated. The discrepancy is most likely due to the extra needed pixel buffers for the CDF 9/7 DWT.

The amount of tri-state buffers obtained from the synthesis reports are shown in Figure 4.12. When synthesized for minimum area the number of tri-state buffers generated for the CDF 9/7 is more than double that for the CDF 5/3 DWT.

Figure 4.11: Sequential Logic Units for CDF 5/3 and CDF 9/7 DWT Synthesized for Maximum Speed/Minimum Area using Ripple-Carry/CLA adders

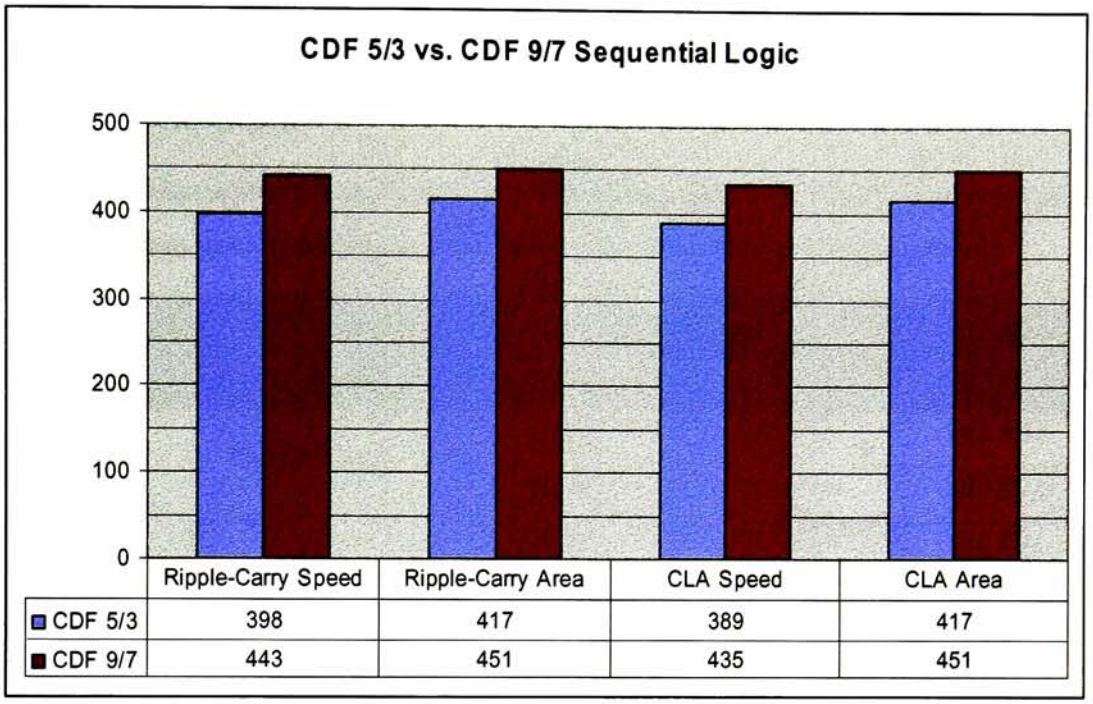
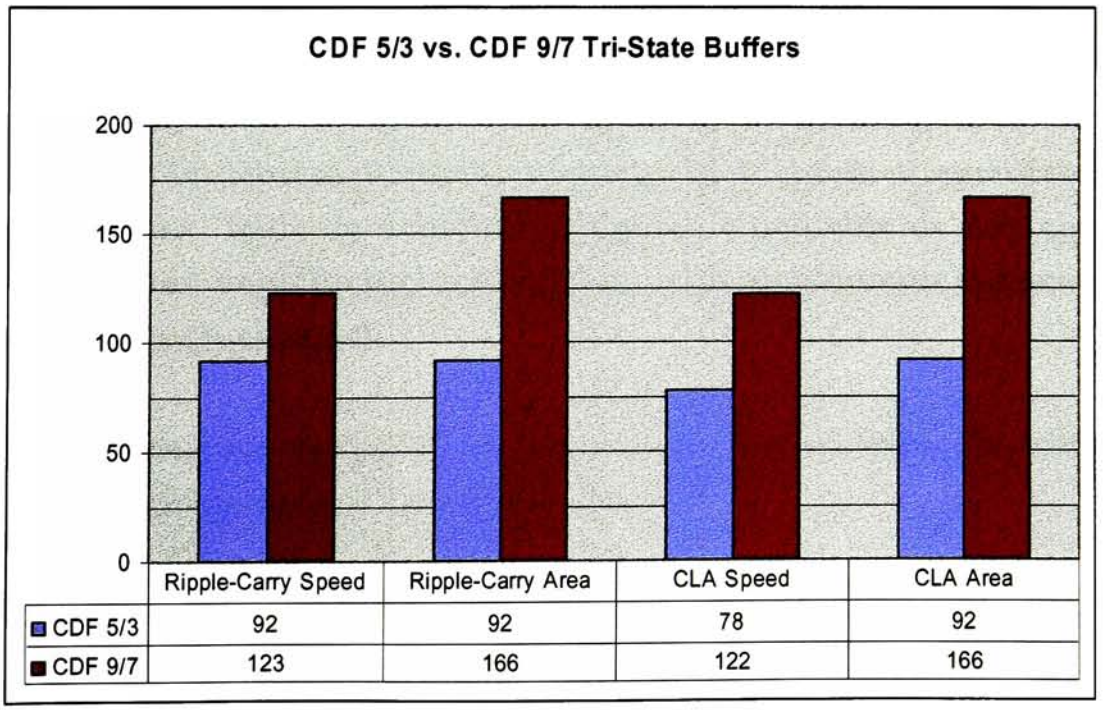


Figure 4.12: Tri-State Buffer Units for CDF 5/3 and CDF 9/7 DWT Synthesized for Maximum Speed/Minimum Area using Ripple-Carry/CLA adders



4.3.2. Hardware Speed

Table 4.2 and Figure 4.13 show the maximum hardware speed determined during synthesis. In all cases the speed of the CDF 5/3 DWT is more than double that of the CDF 9/7. This is most likely due to the long delay paths through the FPT_COEF_MULT units generated for the real-number CDF 9/7 coefficients.

Table 4.2: Maximum Clock Speed for CDF 5/3 and CDF 9/7 DWT Synthesized for Maximum Speed/Minimum Area using Ripple-Carry/CLA adders

Wavelet	Architecture	Optimization	Max Clock Speed (MHz)	Min Clock Period (ns)
CDF 5/3	Ripple-Carry	Speed	6.973	143.420
		Area	3.566	280.392
	CLA	Speed	7.326	136.491
		Area	3.666	272.776
CDF 9/7	Ripple-Carry	Speed	2.218	450.830
		Area	1.533	652.285
	CLA	Speed	2.383	419.725
		Area	1.547	646.290

Using the timing results from earlier, an image of size 512x512 requires 512 x 512 x 14.5 = 3,801,088 clock cycles for a 1-level DWT. Running at the maximum clock speed of 7.326 MHz this would yield one transformed 512x512 image every 0.519 seconds or approximately two images per second. Improving a hardware performance to 100 MHz via more aggressive synthesis constraints and/or custom circuit design would improve performance to approximately 26 transformed images per second.

As mentioned earlier only the CDF 5/3 was fully synthesizable to the FPGA. The information below is for the final synthesized CDF 5/3 DWT core.

Figure 4.13: Maximum Clock Speed for CDF 5/3 and CDF 9/7 DWT Synthesized for Maximum Speed/Minimum Area using Ripple-Carry/CLA adders

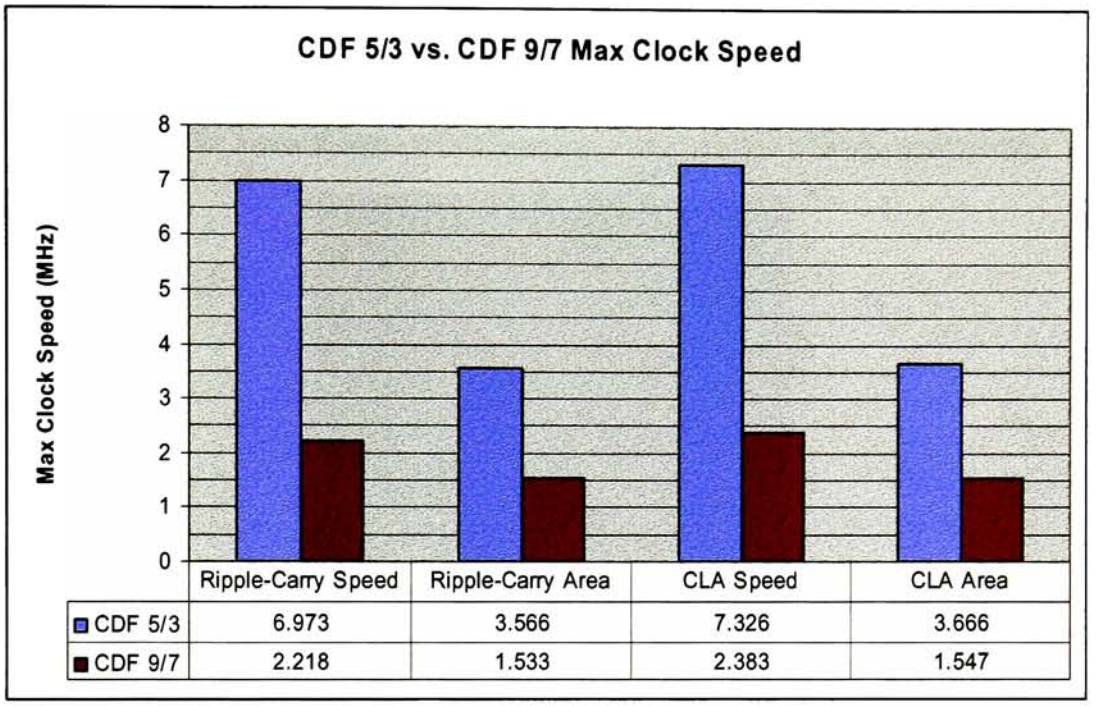


Table 4.3: Maximum Clock Speed and Equivalent Gate Count for Hardware CDF 5/3 DWT Synthesized for Maximum Speed/Minimum Area using Ripple-Carry/CLA adders

Architecture	Optimization	Max Clock Speed (MHz)	Min Clock Period (ns)	Equivalent Gate Count
Ripple-Carry	Speed	2.185	457.711	16,410
	Area	1.561	640.595	15,918
CLA	Speed	2.185	457.704	17,628
	Area	1.676	596.674	16,656

The CLA version is the fastest running up to 2.185 MHz. The ripple-carry version is the smallest with only a 15,918 equivalent gate count using only about 5% of the available hardware resources of the Xilinx FPGA. All synthesized designs have a power consumption rating of 24 mW using voltage (Vcc) of 2.5 V and current (I) of 10 mA.

4.4. Denoising Results

MATLAB is used to add three different types of noise to the 512x512 pixel 8-bit grayscale Lena image: Gaussian, Salt & Pepper, and Speckle. Each of these images are shown in Figures 4.14, 4.15, and 4.16.

Figure 4.14: Lena Image with Gaussian Noise Added (PSNR = 20.23 dB)

Figure 4.15: Lena Image with Salt & Pepper Noise Added (PSNR = 18.17 dB)

Figure 4.16: Lena Image with Speckle Noise Added (PSNR = 21.19 dB)

A MATLAB denoising module is used to perform denoising on the noisy images using different variation of the hard and soft wavelet thresholding denoising techniques mentioned earlier. All of these techniques are fully in-place no extra pixel buffers. Combined with the DWT core this denoising technique requires no intermediate pixel buffering reducing the memory requirements. In addition the denoising techniques applied below are non-adaptive (independent of the pixel values) therefore the execution time is constant regardless of the amount of denoising being applied.

Using MATLAB scripts three different denoising techniques are applied to each of the three noisy images using both CDF 5/3 and CDF 9/7 DWT and hard and soft thresholding. The first technique uses a constant threshold value across scales. The second technique uses a threshold the decreases by a constant value (linear decay) across scales. The third technique uses a threshold that decreases by a threshold value that is decreased by a specific factor (exponential decay) across scales. Increasing the threshold across scales does not improve denoising performance in any way and therefore is not considered.

The following sections analyze graphs from Appendix B of the PSNR and Percent Similarity between the original clean image and the denoised image as the threshold values, thresholding factors and levels of decomposition changes. The "cleanest" images from each test determined by PSNR and percent difference benchmarks are also shown in Appendix B.

4.4.1. Constant

Image denoising using a constant threshold is performed using all combinations of threshold values between 1 and 255 and 1 to 9 levels of DWT decomposition for the CDF 5/3 and 1 to 4 levels of decomposition for the CDF 9/7. Graphs and images with the results for the denoising are found in Appendix B.

4.4.2. Variable – Linear

Image denoising using a linearly variable threshold is performed all combinations of threshold values starting between 10 and 130 (incrementing by 10) varying by all factors between 10 and 250 (incrementing by 10) for DWT decompositions levels 2 and 3. Results for decomposition levels greater 3 are not shown below since they did not provide improved denoising performance. An anomaly in the results showing a steep decline in denoising performance at certain thresholds followed by a steady rise is a result of an error in the MATLAB code where the variable threshold falls below zero and becomes a negative value increasing in magnitude.

4.4.3. Variable – Exponential

Image denoising using a linearly variable threshold is performed all combinations of threshold values starting between 10 and 130 (incrementing by 10) varying by all factors between 2 and 15 for DWT decompositions levels 2 and 3. Results for decomposition levels greater 3 are not shown below since they did not provide improved denoising performance.

4.4.4. Denoising Analysis

The cleanest images for each of the three types of noise using all types of denoising are shown in Figure 4.17, 4.18 and 4.19. For all types of noise it appears that denoising via soft thresholding with an exponentially decaying threshold across 3-4 scales proved to be optimal

Figure 4.17: Best Wavelet Denoising Results for Lena Image with Gaussian Noise

CDF 5/3
Exponential
Soft Thresholding
Level: 4
Threshold: 110
Factor: 4
PSNR(dB): 27.4452

CDF 5/3
Exponential
Soft Thresholding
Level: 4
Threshold: 150
Factor: 4
Pct Diff(%): 3.0488

Figure 4.18: Best Wavelet Denoising Results for Lena Image with Salt & Pepper Noise

CDF 9/7
Exponential
Soft Thresholding
Level: 3
Threshold: 250
Factor: 6
PSNR(dB): 26.0508

CDF 5/3
Linear
Soft Thresholding
Level: 3
Threshold: 250
Factor: 120
Pct Diff(%): 3.5235

Figure 4.19: Best Wavelet Denoising Results for Lena Image with Speckle Noise

CDF 5/3
Exponential
Soft Thresholding
Level: 4
Threshold: 130
Factor: 5
PSNR(dB): 27.6880

CDF 5/3
Exponential
Soft Thresholding
Level: 4
Threshold: 160
Factor: 5
Pct Diff(%): 2.9312

Chapter 5. Conclusion

A flexible hardware architecture for performing the DWT on a digital image was presented in this thesis. This architecture uses a variation of the lifting scheme technique that provides significant advantages over the both the standard lifting scheme-based DWT and convolution-based DWT such as smaller memory requirements, fixed-point arithmetic instead of more costly floating-point, and less arithmetic computations. In addition the architecture is flexible in that it can be configured to perform many different variations of the DWT.

The CDF 5/3 hardware produces identical results to its theoretical MATLAB model. The fixed-point CDF 9/7 deviates very slightly from its floating-point MATLAB model with a ~59dB PSNR deviation for nine levels of DWT decomposition. The execution time for performing both DWTs is nearly identical at ~14 clock cycles per image pixel for one level of DWT decomposition. The hardware area generated for the CDF 5/3 is ~16,000 gates using only 5% of the Xilinx FPGA hardware area, 2.185 MHz maximum clock speed and 24 mW power consumption. The simple wavelet image denoising techniques resulted in cleaned images up to ~27 PSNR.

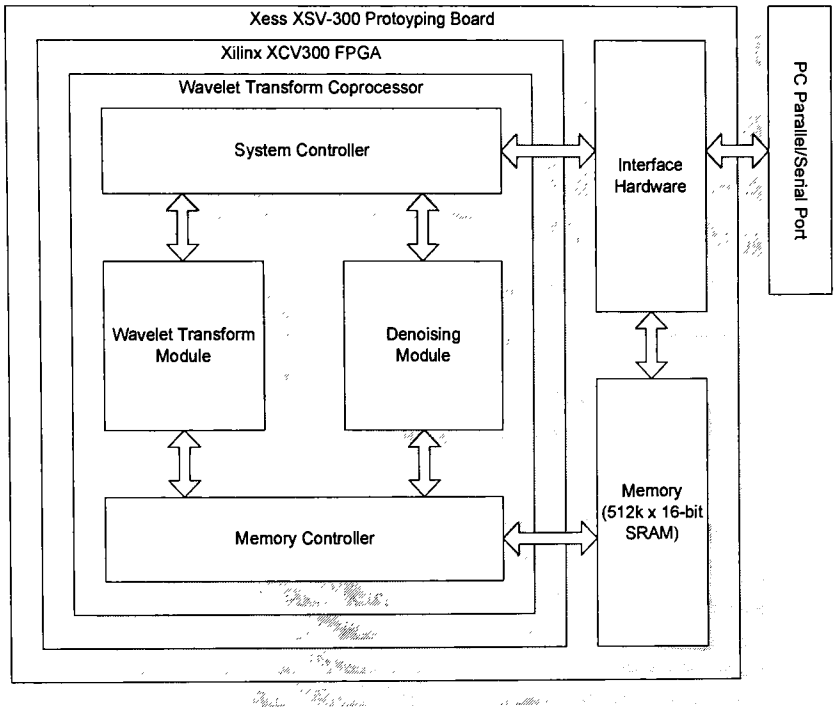
Various recommendations for future work in the area of the discrete wavelet transform related to the DWT core designed for this thesis are presented below.

5.1. Recommendations for Future Work

5.1.1. Hardware Implementation of Denoising Algorithm

The software wavelet denoising algorithm used in this thesis can be ported to hardware, such as the prototyping board used in this thesis to create a DWT-based image denoising processor. The denoising algorithms would simply need to be ported from software to hardware. Figure 5.1 show an example module diagram of a image denoising processor implemented on the XSV-300 FPGA prototyping board.

Figure 5.1: Module Diagram of Hardware Implementation of Wavelet Denoising Processor



5.1.2. Integration of DWT Core Into Image Coprocessor/DSP

The DWT core can be integrated into a digital signal processor used to perform digital image processing on an image as shown in Figure 6.1. The versatility of the DWT core would allow a custom ASIC designer to tailor the DWT to the particular application. For example, a JPEG2000 core would use the CDF 5/3 and CDF 9/7 DWT cores.

Figure 5.2: Module Diagram of Generic Hardware Implementation of Wavelet –Based Image Processor

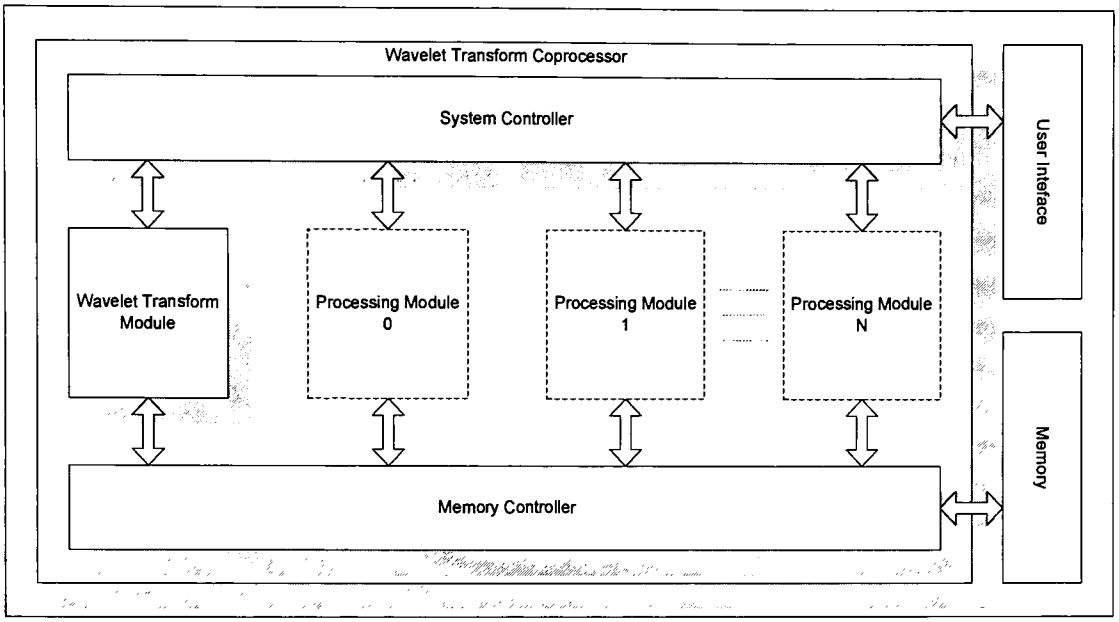
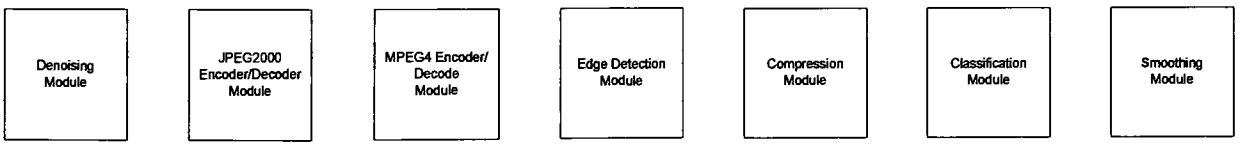


Figure 6.2 shows modules that could be incorporated into one or more DWT-based DSPs.

Figure 5.3: Possible Image Processing Modules



5.1.3. 1-D DWT

The DWT core can be easily configured to perform the DWT on 1-D signals such as audio.

5.1.4. Pipelined Architecture

A pipelined version of the DWT core can be implemented to reduce propagation delay through the lift_alu_group unit. Extra pixel buffers need to be inserted in between the cascaded lift alu units to reduce the critical path and thus increase throughput and the datapath needs to be slightly modified.

5.1.5. Scalable Architecture and Parallel Processing

The DWT core can be modified to either manually or automatically detect the presence of other DWT cores and in doing so coordinate efforts to perform a parallel DWT on a single image. The DWT is a highly parallelizable algorithm since each row is processes independently of each other. Task scheduling algorithms for coordinating multiple DWT cores can range from simple to complex. A simple example is when two DWT cores are present. One core could process the even lines of an image while the other core processes the odd lines of an image. Given sufficient memory bandwidth this configuration would provide a speed up of two. The addition of other DWT cores would continue to provide a linear speed up until the point where DWT cores are idle from lack of work.

More complex task scheduling can be performed to improve performance. For example, a single row of an image can be processed by multiple DWT cores making sure they do not overlap one another.

An even more sophisticated parallelized algorithm would be to process both rows and columns simultaneously to provide a 2-D line-based DWT. Certain DWT cores can begin processing columns as the portions of the necessary rows are gradually processed by row processors. Images such as satellite photos that continuously scan in one direction and therefore are unbounded in one dimension normally would require the image to be broken into smaller, bounded tiles to be processed separately. A 2-D line-based DWT would perform the 2-D DWT on an image progressively, only buffering the minimum number of lines needed. Not only does this allow for the complete 2-D transform to be performed on an unbounded image without tiling, but it also allows for significant memory savings as only a few rows of the image are buffered rather than the entire image. Also, since portions of the image are fully transformed much sooner than the standard row-column DWT the information is available much sooner. This would speed up performance in such processes as image compression since the quantizer would be able to start processing the transformed image before it is fully transformed. This scalable architecture would be a very simple way to improve DWT performance given addition hardware real estate. Multiple DWT cores can be implemented on a single chip if extra space is available or DWT cores from different chips can interface with one another.

The intelligence of each DWT core can be modified as well. For example, each core may be moderately intelligent containing the control logic to detect the presence of other DWT cores and then coordinate efforts amongst themselves. The complexity of the task scheduling in this scenario is limited. A more advanced system would have possibly one very intelligent stand alone "master" task scheduler which would coordinate the efforts between all the less intelligent "slave" DWT cores. The slave DWT cores would have no knowledge of each other and simply take direction from the master task scheduler.

5.1.6. Incorporation of More Than One DWT

Given the flexible nature of the DWT core more than one wavelet could be easily integrated for more functionality. Almost all of the hardware can be shared between two or more wavelet transforms on the same core reducing the overhead. The hardware would only be slightly larger than that generated for the largest DWT if it were stand-alone. The sliding window buffer would need to be large enough to accompany the most complex of the DWTs and the smaller wavelet would use only a portion of that buffer. The simplest implementation would use different lift_alu_group units for each wavelet, however, a more sophisticated design would reuse the ALUs. Perhaps the ALUs would be dynamically configurable based on a lookup table containing the lifting and scaling step coefficients for each DWT. Provisions for selecting between the multiple DWTs would be necessary. Most of the other operations such as reordering and control are the same regardless of the wavelet. The VHDL code would be modified to add yet another dimension describing not just one but multiple wavelets to be integrated onto a single core.

This can be taken further by allowing the hardware to dynamically configure its hardware at runtime to perform a specific wavelet. This would be useful for adaptive DWT when the requirements of the DWT change at run time. The hardware overhead for such an implementation would be significant.

5.1.7. Incorporation of Convolution and Standard Lifting

The DWT core currently uses a modified version of the lifting scheme to perform the DWT using only a single pass of a row of pixels. This DWT core can be modified to perform the DWT using traditional convolution and/or the standard lifting scheme in addition to its current modified lifting scheme technique if so desired.

The convolution implementation requires a convolution kernel buffer for the high and low pass filters. It would be possible to add extra control logic and ALUs to use the current DWT core's sliding window buffer as both the high and low pass filters. The ALUs would need to be reconfigurable at runtime to change between the high/low pass and forward/inverse convolution operations. Also a line buffer would be needed to store the results of the convolution since it is not in-place. The convolution can be specifically tailored to perform the DWT in single pass and in-place by slightly modifying the DWT core control logic. The disadvantage of either convolution implementation is still number of costly arithmetic operations and hardware.

The standard lifting scheme is a much more simplified version of the current DWT that would require less hardware than the current DWT core. The tradeoff is speed as the standard implementation requires multiple passes of every row of pixels rather than just one. The sliding window buffer would be eliminated and replaced by a simple three-pixel buffer. Unlike the DWT core this buffer is the same size regardless of the wavelet being implemented. Only one ALU unit is required rather than an entire lift_alu_group. This ALU needs to be reconfigured for each lifting step. The control logic would now be required to perform multiple passes of a row of pixels reconfiguring the ALU with the appropriate lift coefficient. This implementation, although slower than the DWT core, has more potential for being dynamically reconfigurable in hardware. A lookup table containing the number of lifting steps and the lifting coefficients is all that is needed to configure the hardware at runtime to perform any wavelet.

5.1.8. Software VHDL Code Generator

The current VHDL source for the DWT core uses many generate statements used to dynamically create complex hardware at synthesis time from the user defined constants describing the

wavelet. This may pose a problem for less sophisticated synthesis tools such as XST that cannot generate hardware in multiple dimensions.

A possible solution is to use a software application to generate the desired VHDL source code. Rather than the burden of dynamically generating hardware falling solely on the VHDL source the source code generator application could be used to perform the manual synthesis for the user and write VHDL code that is less or not dependent on complex generate statements. This would guarantee and improve synthesis by less sophisticated synthesis tools.

This software application can be taken even further by performing lifting step extraction for the user. The current DWT requires that the appropriate lifting and scaling steps to perform the DWT are extracted ahead of time from an external source and hardcoded into the VHDL source prior to synthesis. A software package could be developed that performs automatic lifting step extraction directly from the high and low pass filter coefficient descriptions of the wavelet to provide to the VHDL source automatically. This lifting step extractor can either be stand alone or more effectively combined with the source code generator to fully automate the implementation of the hardware directly from the wavelet description with little or no knowledge of VHDL. Lifting step extraction could be implemented in the hardware at a very high cost if adaptive DWTs are desired.

5.1.9. Improve Hardware Performance

A more sophisticated synthesis tool, such as Synopsys, could be used to generate a more efficient hardware design than is currently produced by XST. Synopsys could be used to generate hardware to run as a faster clock speed and/or use less hardware resources. Xilinx technology libraries would be needed to synthesize to the Xilinx FPGA. Perhaps Synopsys would be able to completely synthesize the VHDL description of the DWT core without manual synthesis intervention from the user.

The VHDL source code could be written at a higher level of abstraction to make better use of inference engines of the synthesis tools. Currently the VHDL code describes simple logic units such as adders and multipliers at a very low structural level taking the burden off of the synthesis

tool. The VHDL code could instead use the arithmetic operators '+' and '*' to leave the hardware inference up to the synthesis tool to generate possibly more efficient hardware for the target technology.

More aggressive full-custom design techniques and timing can be applied to the DWT core to improve hardware performance beyond what it is currently. The VHDL code is at a very low structural level and therefore can be easily modified by a full-custom hardware designer to improve performance on ASICs. The current 14.5 clock cycle per pixel execution time can be reduced further by improving the control mechanism to reduce wasted clock cycles. Reordering can be performed after all the lifting is performed for all levels of DWT rather than after each level to reduce memory bandwidth, however, hardware complexity increases.

References

[1] R. Polikar. "The Wavelet Tutorial Part I Fundamental Concepts and an Overview of the Wavelet Theory". Iowa State University. Internet URL: "http://engineering.rowan.edu/~polikar/WAVELETS/WTtutorial.html".

[2] R. Polikar. "The Wavelet Tutorial Part II Fundamentals: The Fourier Transform and the Short Term Fourier Transform". Iowa State University. Internet URL: "http://engineering.rowan.edu/~polikar/WAVELETS/WTtutorial.html".

[3] C. Valens. "A Really Friendly Guide to Wavelets". 1999.

[4] R. Gonzalez and R. Woods. Digital Image Processing. Addison-Wesley Publishing Company, New York, 1992.

[5] R. Polikar. "The Wavelet Tutorial Part III – Multiresolution Analysis Concepts & and the Continuous Wavelet Transform ". Iowa State University. Internet URL: "http://engineering.rowan.edu/~polikar/WAVELETS/WTtutorial.html".

[6] R. Polikar. "The Wavelet Tutorial Part IV Multiresolution Analysis: The Discrete Wavelet Transform". Iowa State University. Internet URL: "http://engineering.rowan.edu/~polikar/WAVELETS/WTtutorial.html".

[7] D. Taubman and M. Marcellin. JPEG2000 Image Compression Fundamentals, Standards and Practice. Kluwer Academic Publishers, Norwell, MA, 2002.

[8] W. Sweldens and P. Schroder. "Building Your Own Wavelets at Home". In *Wavelets in Computer Graphics*, pp. 15-87, 1996.

[9] W. Sweldens. "The Lifting Scheme: A New Philosophy in Biorthogonal Wavelet Constructions". In *Proc. SPIE*, vol. 2569, pp. 68-79, 1995.

[10] I. Daubechies and W. Sweldens. "Factoring Wavelet Transforms into Lifting Steps". In *J. Fourier Anal. Appl.*, vol. 4, n. 3, pp. 247-269, 1998.

[11] J. K. Romberg, C. Hyeokho, and R. G. Baraniuk, "Shift-invariant denoising using wavelet-dornain hidden Markov trees," *1999 Conference Record of the Thirty-Third Asilomarf Conference on Signals, Systems, and Computers*, vol. 2, pp. 1277-1281. Oct. 1999.

[12] C. Taswell. "The What, How, and Why of Wavelet Shrinkage Denoising". In *IEEE: Computing in Science & Engineering*, pp. 12-19, May/June 2000.

[13] Y. Hawwar, A. Reza, and R. Turney. "Filtering (Denoising) in the Wavelet Transform Domain". University of Wisconsin-Milwaukee: Dept. of EE and CS and Xilinx, Inc.: CORE Solutions Group, San Jose, CA.

[14] J. Qian. "Denoising by Wavelet Transform". Rice University: Department of EE.

[15] J. Liu and P. Moulin. "Image Denoising Based on Scale-Space Mixture Modeling of Wavelet Coefficients". University of Illinois: Beckman Institute and ECE Department, IEEE, 1999.

[16] S. Subhasis. "Image Compression – from DCT to Wavelets: A Review". ACM, Inc. 2000. Internet URL: "http://www.acm.org/corssroads/xrds6-3/sahaimgcoding.html".

[17] A. Skodras, C. Christopoulos, and T. Ebrahimi. "The JPEG 2000 Still Image Compression Standard". *IEEE Signal Processing Mag.*, vol. 18, pp. 36-58, Sept. 2001.

[18] Mathworks, Inc Website: Internet URL: "http://www.matlab.com".

[19] Analog Devices, Inc. Website: Internet URL: "http://www.analogdevices.com".

[20] CAST, Inc. Website. Internet URL: "http://www.cast-inc.com".

[21] A. Grzeszczak, M. K. Mandal, S. Panchanathan, and T. Yeap. "VLSI Implementation of Discrete Wavelet Transform". *IEEE Transactions on VLSI Systems.*, vol. 4, pp. 421-433, Dec. 1996.

[22] Y. Kim, K Jun, K Rhee. "FPGA Implementation of Subband Image Encoder using Discrete Wavelet Transform". Inha Technical Junior College: Department of EE, Chosun University: School of Electronics and Info-Comm. Eng.. IEEE, 1999.

[23] K. Andra, C. Chakrabarti, and T. Acharya. "A VLSI Architecture for Lifting-Based Forward and Inverse Wavelet Transform". In *IEEE Transactions on Signal Processing*, vol. 50, no. 4. April 2002.

[24] J. Jou, Y. Shiau, and C. Liu. "Efficient VLSI Architectures for the Biorthogonal Wavelet Transform by Filter Bank and Lifting Scheme". National Cheng Kung University: Dept. of EE. IEEE, 2001.

[25] M. Martina, G. Masera, G. Piccinini, and M. Zamboni. "A VLSI Architectures for IWT (Integer Wavelet Transform)". In *Proc. 43rd IEEE Midwest Symposium on Circuits and Systems*. Lansing, MI. Aug. 2000.

[26] M. Grangetto, M. Martina, G. Masera, G. Piccinini, F. Vacca, and M. Zamboni. "FPGA Power Efficient Inverse Lifting Wavelet IP". Torino, Italy. IEEE, 2001.

[27] P. McCanny, S. Masud, and J. McCanny. "An Efficient Architecture for the 2-D Biorthogonal Discrete Wavelet Transform". Belfast, N. Ireland. IEEE, 2001.

[28] C. Lian, K. Chen, H. Chen, and L. Chen, "Lifting Based Discrete Wavelet Transform Architecture for JPEG2000," *The 2001 IEEE International Symposium on Circuits and Systems*, vol. 2, pp. 445-448, May 2001.

[29] V. Splliotopoulos, N. Zervas, Y. Andreopoulos, G. Anagnostopoulos, and C. Goutis. "Quantization Effect on VLSI Implementations for the 9/7 DWT Filters". University of Patras: Dept. of ECE. IEEE, 2001.

[30] W. Chang, Y. Lee, W. Peng, and C. Lee. "A Line-Based, Memory Efficient and Programmable Architecture for 2D DWT using Lifting Scheme". In *Proceedings of IEEE ISCAS 2001*. IEEE, 2001.

[31] C. Huang, P. Tseng, and L. Chen. "Efficient VLSI Architectures of Lifting-Based Discrete Wavelet Transform by Systematic Design Method". National Taiwan University: Dept. of EE. IEEE, 2002.

[32] G. Savaton, E. Casseau, and E. Martin. "High Level Design and Synthesis of a Discrete Wavelet Transform Virtual Component for Image Compression". In *IP Based Design 2000*, Dec. 2000.

[33] Y. Fisher et al. Fractal Image Compression. Spring Verlag, 1995.

Appendix

Appendix A: DWT Images

Note: The following results can be found on accompanying data CD

Lena Images:

Lena Image CDF 5/3 Forward DWT using Convolution/Lifting Scheme (1-9 Levels)
Lena Image CDF 9/7 Forward DWT using Convolution (1-7 Levels)
Lena Image CDF 9/7 Forward DWT using Lifting Scheme Actual (1-9 Levels)
Lena Image CDF 9/7 Forward DWT using Lifting Scheme Approximate (1-9 Levels)

Rapper Images:

Rapper Image CDF 5/3 Forward DWT using Lifting Scheme (1-6 Levels)
Rapper Image CDF 9/7 Forward DWT using Lifting Scheme (1-6 Levels)

Appendix B: Denoising Results

Note: The following results can be found on accompanying data CD

Constant/Hard Thresholding:

Constant Hard Thresholding CDF 5/3 Gaussian
Constant Hard Thresholding CDF 5/3 Salt & Pepper
Constant Hard Thresholding CDF 5/3 Speckle
Constant Hard Thresholding CDF 9/7 Gaussian
Constant Hard Thresholding CDF 9/7 Salt & Pepper
Constant Hard Thresholding CDF 9/7 Speckle

Constant/Soft Thresholding:

Constant Soft Thresholding CDF 5/3 Gaussian
Constant Soft Thresholding CDF 5/3 Salt & Pepper
Constant Soft Thresholding CDF 5/3 Speckle
Constant Soft Thresholding CDF 9/7 Gaussian
Constant Soft Thresholding CDF 9/7 Salt & Pepper
Constant Soft Thresholding CDF 9/7 Speckle

Variable - Linear Thresholding:

Variable – Linear CDF 5/3 Gaussian
Variable – Linear CDF 5/3 Salt & Pepper
Variable – Linear CDF 5/3 Speckle
Variable – Linear CDF 9/7 Gaussian
Variable – Linear CDF 9/7 Salt & Pepper
Variable – Linear CDF 9/7 Speckle

Variable - Exponential Thresholding:

Variable – Exponential CDF 5/3 Gaussian
Variable – Exponential CDF 5/3 Salt & Pepper
Variable – Exponential CDF 5/3 Speckle
Variable – Exponential CDF 9/7 Gaussian
Variable – Exponential CDF 9/7 Salt & Pepper
Variable – Exponential CDF 9/7 Speckle

Appendix C: MATLAB Source Code

Note: The following code can be found on accompanying data CD

DWT Core:

wavelet_2d.m
denoise.m

Test Scripts:

dwt_tests.m
denoise_tests.m

Utility Files:

image_read.m
image_write.m
sym_ext.m
compose_image.m
decompose_image.m
mult_fwd_inv.m
print_pixels.m
eq_subbands.m
pct_diff.m
pct_diff2.m
psnr.m
psnr2.m
image_diff.m
image_compare.m
image_check.m
image_histeq.m
wavelet_coef_stats.m

Appendix D: VHDL Source Code

Note: The following code can be found on accompanying data CD

DWT Core:

wavelet_pkg.vhd
wavelet_2d.vhd
memory_controller.vhd
multi_lift_reorder_2d.vhd
multi_lift_reorder_2d_ctrl.vhd
lift_reorder_2d.vhd
lift_reorder_2d_ctrl.vhd
multi_lift_reorder_1d.vhd
multi_lift_reorder_2d_ctrl.vhd
lift_reorder_1d.vhd
lift_reorder_1d_ctrl.vhd
lift_1d.vhd
lift_1d_ctrl.vhd
sliding_window.vhd
reorder_adr_calc.vhd
reorder_status_unit.vhd
lift_alu_group.vhd
lift_alu.vhd
scale_alu.vhd
fpt_coef_mult.vhd
nbit_add_sub
fa.vhd
nbit_array_multiplier.vhd
nbit_comparator.vhd
nbit_inc_dec.vhd
nbit_shift_register.vhd
nbit_barrel_shifter.vhd
shift_counter.vhd
nbit_decoder.vhd
nbit_encoder.vhd

Software Simulation:

sim_test.vhd
sram_model.vhd

FPGA:

fpga_test.vhd
lift_alu1.vhd
lift_alu2.vhd
lift_alu_group.vhd (modified)
wavelet.pkg (modified)

Appendix E: C Source Code

Note: The following code can be found on accompanying data CD

pgm2xes.c
xes2pgm.c

LH3151 GU23 6147 CD